digital

OpenVMS Delta/XDelta Debugger Manual

OpenVMS

Part Number: AA–PWCAA–TE

OpenVMS Delta/XDelta Debugger Manual

Order Number: AA–PWCAA–TE

May 1993

This manual describes the OpenVMS Delta and XDelta debuggers. OpenVMS Delta is used to debug programs that run in privileged processor mode at interrupt priority level 0. OpenVMS XDelta is used to debug programs that run at an elevated interrupt priority level.

Revision/Update Information: This manual supersedes the *VMS Delta/XDelta Utility Manual*.

Software Version: OpenVMS VAX Version 6.0
OpenVMS AXP Version 1.5

Digital Equipment Corporation
Maynard, Massachusetts

May 1993

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may be used or copied only in accordance with the terms of such license.

No responsibility is assumed for the use or reliability of software on equipment that is not supplied by Digital Equipment Corporation or its affiliated companies.

© Digital Equipment Corporation 1993.

All Rights Reserved.

The postpaid Reader's Comments forms at the end of this document request your critical evaluation to assist in preparing future documentation.

The following are trademarks of Digital Equipment Corporation: Alpha AXP, AXP, Bookreader, BI, CI, Digital, HSC, MicroVAX, TU58, OpenVMS, UDA, VAX, VAXBI, VAX DOCUMENT, VAXft, VAX MACRO, VAXstation, VMS, VT, the AXP logo, and the DIGITAL logo.

All other trademarks and registered trademarks are the property of their respective holders.

ZK4540

This document was prepared using VAX DOCUMENT, Version 2.1.

Contents

Preface vii

1 Invoking, Exiting, and Setting Breakpoints

1.1 Overview of the Delta and XDelta Debuggers 1–1
1.2 Privileges Required for Running DELTA 1–1
1.3 Guidelines for Using XDELTA 1–2
1.4 Invoking DELTA 1–2
1.5 Exiting from DELTA 1–2
1.6 Invoking XDELTA 1–2
1.6.1 Booting XDELTA on a VAX 9000 Computer 1–4
1.6.2 Booting XDELTA on a VAX 8530, 8550, 8810 (8700), 8820, 8820-N (8800), 8830, or 8840 Computer 1–5
1.6.3 Booting XDELTA on a VAX 8600 or a VAX 8650 Computer 1–6
1.6.4 Booting XDELTA on a VAX 8200, 8250, 8300, or 8350 Computer 1–6
1.6.5 Booting XDELTA on a VAX 6000 Series Computer 1–7
1.6.6 Booting XDELTA on a VAX 7000 or VAX 10000 Series Computer 1–7
1.6.7 Booting XDELTA on a VAXstation 3100 or MicroVAX 3100 Series or VAXstation 4000 Series Computer 1–8
1.6.8 Booting XDELTA on a VAXft 3000, VAXft-410, VAXft-610, or VAXft-612 Computer 1–9
1.6.9 Booting XDELTA on a MicroVAX 2000, VAXstation 2000, MicroVAX 3300/3400 Series, VAXstation 3520, VAXstation 3540, MicroVAX or VAXstation 3500/3600 Series, MicroVAX 3800/3900, VAX 4000 series, MicroVAX II, or VAX–11/750 Computer 1–9
1.6.10 Booting XDELTA on a VAX–11/780 or a VAX–11/785 Computer 1–9
1.6.11 Booting a VAX–11/730 Computer Using the Console TU58 1–10
1.6.12 Booting XDELTA on AXP Computers 1–11
1.7 Requesting an Interrupt 1–11
1.7.1 Requesting Interrupts on VAX Computers 1–11
1.7.2 Requesting Interrupts on AXP Computers 1–12
1.8 Accessing the Initial Breakpoint 1–13
1.9 Proceeding from Initial XDELTA Breakpoints 1–13
1.10 Exiting from XDELTA 1–14

2 DELTA and XDELTA Symbols and Expressions

2.1 Symbols Supplied by DELTA and XDELTA 2–1
2.2 Forming Numeric Expressions 2–2

3 Debugging Programs

3.1 Referencing Addresses 3–1
3.1.1 Referencing Addresses (VAX Only) 3–2
3.1.2 Referencing Addresses (AXP Only) 3–4
3.2 Referencing Registers (VAX Only) 3–6
3.3 Referencing Registers (AXP Only) 3–6
3.4 Interpreting the Error Message 3–7
3.5 Debugging Kernel Mode Code Under Certain Conditions 3–7
3.5.1 Setup Required (VAX Only) 3–7
3.5.2 Setup Required (AXP Only) 3–7
3.5.3 Accessing XDELTA 3–8
3.6 Debugging an Installed, Protected, Shareable Image 3–8
3.7 Using XDELTA on Multiprocessor Computers 3–9
3.8 Debugging Code When Single-Stepping Fails (AXP Only) 3–9
3.9 Debugging Code That Does Not Match the Compiler Listings (AXP Only) 3–10

4 DELTA/XDELTA Commands

4.1 Command Usage Summary 4–1
[(Set Display Mode) 4–3
/ (Open Location and Display Contents in Prevailing Width Mode) 4–4
! (Open Location and Display Contents in Instruction Mode) 4–7
LINEFEED (Close Current Location, Open Next) 4–9
ESC (Open Location and Display Previous Location) 4–11
TAB (Open Location and Display Indirect Location) 4–12
" (Open Location and Display Contents in ASCII) 4–13
RETURN (Close Current Location) 4–15
;B (Breakpoint) 4–16
;P (Proceed from Breakpoint) 4–19
;G (Go) 4–20
S (Step Instruction) 4–21
O (Step Instruction over Subroutine) 4–23
' (Deposit ASCII String) 4–26
;E (Execute Command String) 4–27
;X (Load Base Register) 4–29
= (Display Value of Expression) 4–31
;M (Set All Processes Writable) 4–32
;L (List Names and Locations of Loaded Executive Images) 4–33
;Q (Validate Queue) (AXP Only) 4–35
;C (Force the system to bugcheck and crash) (AXP Only) 4–36
;W (List Name and Location of a Single Loaded Image) (AXP Only) 4–37
;I (List Information About the Current Main Image and Its Shareable Images)(AXP Only) 4–39
;H (Video Terminal Display Command) (AXP Only) 4–41
\ *string*\ (Immediate mode text display command) (AXP Only) 4–42
EXIT (Exit from DELTA Debugging Session) 4–43

A Sample DELTA Debug Session on VAX

B Sample DELTA Debug Session on AXP

Index

Examples

A–1 Program for Getting LOGINTIMs A–1
A–2 LOGINTIM Program .Map File A–2
A–3 DELTA Debugging Session Example A–3
B–1 Listing File for LOG: C Source Code B–1
B–2 Listing File for LOG: Machine Code B–3
B–3 .MAP File for the Sample Program B–7
B–4 DELTA Debugging Session of the Sample Program B–7

Tables

1–1 Boot Command Qualifier Values 1–4
1–2 BOOT Command Qualifiers for the VAX 9000 Computer 1–4
1–3 Boot Command Qualifiers for the VAX 6000 Series 1–7
1–4 Boot Command Qualifiers (Local and HSC Disks) 1–8
2–1 DELTA/XDELTA Symbols 2–1
2–2 Arithmetic Operators 2–3
4–1 DELTA/XDELTA Command Summary 4–1
4–2 DELTA/XDELTA Command Summary (AXP Only) 4–2

Preface

Intended Audience

This manual is written for programmers who debug system code for device drivers and other images that execute in privileged processor-access modes or at an elevated interrupt priority level (IPL).

Document Structure

This manual consists of the following chapters and appendixes:

- Chapter 1—Provides an overview and descriptions for invoking, exiting, and setting breakpoints.
- Chapter 2—Describes the DELTA and XDELTA symbols and expressions.
- Chapter 3—Describes how to debug programs using DELTA and XDELTA.
- Chapter 4—Describes the DELTA and XDELTA commands, including format, parameters, and examples.
- Appendix A—Describes an OpenVMS VAX debugging session using DELTA.
- Appendix B—Describes an OpenVMS AXP debugging session using DELTA.

Associated Documents

This manual refers to several documents that contain the primary descriptions of topics discussed in this manual. The following table lists the topics and those documents.

Topic	Document
Accessing OpenVMS VAX through a lower priority interrupt level	*OpenVMS System Manager's Manual*
Boot command qualifiers for Volume Shadowing Phase I	*VAX Volume Shadowing Manual*
Boot command qualifiers for Volume Shadowing Phase II	*Volume Shadowing for OpenVMS*
Device name parameters	*OpenVMS System Manager's Manual*
Editing the boot command procedure for certain VAX processors	*OpenVMS VAX Upgrade and Installation Supplement: VAX 8530, 8550, 8810 (8700), and 8820–N (8800)*
IPRs for OpenVMS AXP	*Alpha Architecture Reference Manual*
IPRs for OpenVMS VAX	*VAX Architecture Reference Manual*
PALcode opcodes for OpenVMS AXP	*Alpha Architecture Reference Manual*

Topic	Document
SDA commands for OpenVMS VAX	*OpenVMS VAX System Dump Analyzer Utility Manual*
SDA commands for OpenVMS AXP	*OpenVMS AXP System Dump Analyzer Utility Manual*

Conventions

In this manual, every use of OpenVMS AXP means the OpenVMS AXP operating system, every use of OpenVMS VAX means the OpenVMS VAX operating system, and every use of OpenVMS means both the OpenVMS AXP operating system and the OpenVMS VAX operating system.

The following conventions are used to identify information specific to OpenVMS AXP or to OpenVMS VAX:

	The AXP icon denotes the beginning of information specific to OpenVMS AXP.
	The VAX icon denotes the beginning of information specific to OpenVMS VAX.
♦	The diamond symbol denotes the end of a section of information specific to OpenVMS AXP or to OpenVMS VAX.

The following conventions are also used in this manual:

Ctrl/*x*	A sequence such as Ctrl/*x* indicates that you must hold down the key labeled Ctrl while you press another key or a pointing device button.
PF1 *x*	A sequence such as PF1 *x* indicates that you must first press and release the key labeled PF1, then press and release another key or a pointing device button.
Return	In examples, a key name enclosed in a box indicates that you press a key on the keyboard. (In text, a key name is not enclosed in a box.)
. . .	A horizontal ellipsis in examples indicates one of the following possibilities: • Additional optional arguments in a statement have been omitted. • The preceding item or items can be repeated one or more times. • Additional parameters, values, or other information can be entered.
. . .	A vertical ellipsis indicates the omission of items from a code example or command format; the items are omitted because they are not important to the topic being discussed.

( )	In format descriptions, parentheses indicate that, if you choose more than one option, you must enclose the choices in parentheses.
[ ]	In format descriptions, brackets indicate optional elements. You can choose one, none, or all of the options. (Brackets are not optional, however, in the syntax of a directory name in a VMS file specification, or in the syntax of a substring specification in an assignment statement.)
{ }	In format descriptions, braces surround a required choice of options; you must choose one of the options listed.
boldface text	Boldface text represents the introduction of a new term or the name of an argument, an attribute, or a reason. Boldface text is also used to show user input in Bookreader versions of the manual.
italic text	Italic text emphasizes important information, indicates variables, and indicates complete titles of manuals. Italic text also represents information that can vary in system messages (for example, internal error *number*), command lines (for example, /PRODUCER=*name*), and command parameters in text.
UPPERCASE TEXT	Uppercase text indicates a command, the name of a routine, the name of a file, or the abbreviation for a system privilege.
-	A hyphen in code examples indicates that additional arguments to the request are provided on the line that follows.
numbers	All numbers in text are assumed to be decimal, unless otherwise noted. Nondecimal radixes—binary, octal, or hexadecimal—are explicitly indicated.

1

Invoking, Exiting, and Setting Breakpoints

This chapter presents an overview of the Delta and XDelta Debuggers. It then describes the following:

- Privileges required for running DELTA
- Guidelines for using XDELTA
- Invoking and terminating DELTA and XDELTA debugging sessions on OpenVMS VAX and OpenVMS AXP systems
- Booting XDELTA, requesting interrupts, and accessing initial breakpoints on OpenVMS VAX and OpenVMS AXP systems

1.1 Overview of the Delta and XDelta Debuggers

The Delta and XDelta Debuggers are used to monitor the execution of user programs and the OpenVMS operating system. They use the same commands and the same expressions, but they differ in how they operate. DELTA operates as an exception handler in a process context. XDELTA is invoked directly from the hardware SCB vector in a system context.

Because DELTA operates in a process context, use it to debug user-mode programs or programs that execute at interrupt priority level (IPL) 0 in any processor mode. You cannot use DELTA to debug code that executes at an elevated IPL. To debug with DELTA, invoke it from within your process by specifying it as the debugger (as opposed to the symbolic debugger).

Because XDELTA is invoked directly from the hardware SCB vector, it can be used to debug programs executing in any processor mode or at any IPL level. Use it to debug programs that execute at an elevated IPL. Because XDELTA is not process specific, it is not invoked from a process. To debug with XDELTA, you must boot the processor with commands to include XDELTA in memory. XDELTA's existence terminates when you reboot the processor without XDELTA.

1.2 Privileges Required for Running DELTA

No privileges are required to run DELTA to debug a program that runs in user mode. To debug a program that runs in other processor-access modes, the process in which you run the program must have the necessary privileges.

To use the ;M command, your process must have change-mode-to-kernel (CMKRNL) privilege. The ;M command sets all processes writable.

To use the ;L command (List All Loaded Executive Modules), you must have change-mode-to-executive (CMEXEC) privilege.

1.3 Guidelines for Using XDELTA

Because XDELTA is not process specific, privileges are not required.

When using XDELTA, you must use the console terminal. You should run XDELTA only on a standalone system because all breakpoints are handled at IPL 31.

You cannot redirect output from XDELTA. To determine if your system maintains a log file, check your hardware manual. You can produce a log of console sessions by connecting the console port of the system that will boot with XDELTA to the serial port of a LAT server. Then, from another system, use the command SET HOST/LAT/LOG to that LAT port.

1.4 Invoking DELTA

To invoke DELTA, perform the following steps after assembling (or compiling) and linking your program:

1. Define DELTA as the default debugger instead of the symbolic debugger with the following command:

   ```
   $ DEFINE LIB$DEBUG SYS$LIBRARY:DELTA
   ```

2. Use the following RUN command to execute your program:

   ```
   $ RUN/DEBUG MYPROG
   ```

When DELTA begins execution, it displays its name and current version number. DELTA displays the first executable instruction in the program with which it is linked. It displays the address of that instruction, a separator (a slash (/) on VAX and an exclamation point (!) on AXP), and the instruction and its operands.

VAX

On VAX, the name, current version number, and address are displayed as follows:

```
DELTA Version 5.5
address/instruction operands♦
```

AXP

On AXP, the name, current version number, and address are displayed as follows:

```
AlphaVMS DELTA Version 1.0
address!instruction operands♦
```

On AXP and VAX, DELTA is then ready for your commands.

You can redirect output from a DELTA debugging session by assigning DBG$DELTA to the I/O device.

1.5 Exiting from DELTA

To exit from DELTA, type EXIT and press the Return key. When you are in user mode, you exit DELTA and your process remains. When you are in a privileged access mode, your process can be deleted.

1.6 Invoking XDELTA

To invoke XDELTA, perform the following steps:

1. Boot the system using a console command or a command procedure that includes XDELTA.

VAX

2. On VAX, an initial XDELTA breakpoint is taken so that you can set additional breakpoints or examine and change locations in memory. XDELTA displays the following breakpoint message:

```
1 BRK at address
 address/instruction
```

Note

Never clear breakpoint 1 from any code being debugged in XDELTA. If you accidentally clear breakpoint 1 and no other breakpoints are set, you cannot use XDELTA until you reboot again with XDELTA.♦

AXP

On AXP, two initial XDELTA breakpoints are taken so that you can set additional breakpoints or examine and change locations in memory. XDELTA displays the following message for the first breakpoint:

```
BRK 0 at address
 address!instruction♦
```

3. On AXP and VAX, proceed from the initial breakpoint, using the following command:

```
;P Return
```

VAX

On VAX, the procedure for booting the system with XDELTA differs, depending on the model of your system. Each procedure uses commands that include XDELTA in memory and cause the execution of a breakpoint in OpenVMS initialization routines. Execution of the breakpoint instruction transfers program control to a fault handler located in XDELTA.

Some boot procedures require the use of the /R5 qualifier with the boot command. The /R5 qualifier enters a value for a flag that controls the way XDELTA is loaded. The flag is a 32-bit hexadecimal integer loaded into R5 as input to VMB.EXE, the primary boot program. Refer to Table 1–1 for a description of the valid values for this flag.

Note

When you deposit a boot command qualifier value in R5, make sure that any other values you would normally deposit are included. For example, if you were depositing the number of the system root directory from which you were booting and an XDELTA value, R5 would contain both values.♦

AXP

On AXP, the procedure for booting all AXP systems with XDELTA is the same. The flag for specifying boot qualifiers is a 64-bit integer that is passed directly as input to APB.EXE, the primary boot program. Refer to Table 1–1 for a description of the valid values for this flag. ♦

Table 1–1 Boot Command Qualifier Values

Value	Description
0	Normal, nonstop boot (default)
1	Stop in SYSBOOT
2	Include XDELTA, but do not take the initial breakpoint
3	Stop in SYSBOOT, include XDELTA, but do not take the initial breakpoint
6	Include XDELTA, and take the initial breakpoint
7	Include XDELTA, stop in SYSBOOT, and take the initial breakpoint at system initialization

1.6.1 Booting XDELTA on a VAX 9000 Computer

VAX

To boot a VAX 9000 computer with XDELTA, use the BOOT command in the following format:

```
BOOT [/NOSTART][/R5:boot_flags][/R3:shadow][/BI:vaxbi_info][/XMI:xmi_info]
     [/NODE:hsc_info] xxxunit_number)
```

The *xxx* of the *xxxunit_number* boot parameter refers to the abbreviation of the boot command procedure you are using and the *unit_number* refers to the unit number of the drive. For example, if you are booting from an HSC system disk with unit number 30, you would use CI (the abbreviation for CIBOO.CMD) and the unit number 30. The boot parameter would be CI30.

For example, to boot from the default system disk with XDELTA and stop in SYSBOOT for input, enter the following command:

```
>>> B/R5:7
```

Table 1–2 lists the qualifiers to the BOOT command.

Table 1–2 BOOT Command Qualifiers for the VAX 9000 Computer

Qualifier	Function
/NOSTART	Stops the boot operation after the boot command procedure executes. This command lets you deposit values in registers before transferring control to the primary boot program with the START command.
/R5:boot_flags	Deposits a value (in hexadecimal) into R5. This value affects the execution of VMB9AQ.EXE. Using this optional qualifier, you can perform a conversational boot or boot from a different system root. Use this qualifier to boot XDELTA. Refer to Table 1–1 for a description of the valid values for this flag.
/R3:shadow	Specifies shadowing and physical unit boot information. The shadow qualifier and virtual unit number are required only if you are booting a Volume Shadowing Phase I volume. If you plan to boot a Volume Shadowing Phase II volume, you need to set several volume shadowing SYSGEN parameters. For more information, see *Volume Shadowing for OpenVMS*.

(continued on next page)

Table 1–2 (Cont.) BOOT Command Qualifiers for the VAX 9000 Computer

Qualifier	Function
/BI:vaxbi_info	Specifies the BI node number (in hexadecimal) of the node being accessed. Defaults to 0. You do not need to specify this qualifier if you do not access an XBI or XBI-Plus adapter.
/XMI:xmi_info	Specifies the XMI number and the XMI node number (both in hexadecimal) of the node being accessed. Defaults to 0. The hexadecimal number that you specify with this qualifier must be in the format *xy*, where *x* is the XMI number and *y* is the XMI node number.
/NODE:hsc_info	Specifies the CI node number (in hexadecimal) of the HSC being accessed. You can specify a maximum of two CI node numbers (if two HSCs are available). If you deposit two CI node numbers, put the greater number in hexadecimal digits 3 and 2. Put the smaller number in hexadecimal digits 1 and 0. Refer to the upgrade and installation supplement for the VAX 9000 computer for more information about booting when two HSCs are available.

1.6.2 Booting XDELTA on a VAX 8530, 8550, 8810 (8700), 8820, 8820-N (8800), 8830, or 8840 Computer

To boot a VAX 8530, 8550, 8810 (8700), 8820, 8820-N (8800), 8830, or 8840 computer with XDELTA, use the BOOT command in the following format:

B dddn /R5:f

Substitute BCI, BDA, or UDA for *ddd*. Substitute the unit number of the drive holding the system disk for *n*. Refer to Table 1–1 for a description of the valid values for the boot flag (f).

For example, if you have a BCI-controlled system disk with a unit number of 2, the following command would load XDELTA and take the initial breakpoint:

```
>>> B BCI2 /R5:6
```

This command boots the system with BCIBOO.COM, deposits 2 in R3, and deposits 6 in R5.

You can also boot with XDELTA by editing the appropriate *ddd*XDT.COM procedure so that the unit number of the drive is deposited in R3. (See the *OpenVMS VAX Upgrade and Installation Supplement: VAX 8530, 8550, 8810 (8700), and 8820–N (8800)* for details about editing the command procedure.) Then you can enter the BOOT command in the following format:

@dddXDT

Substitute BCI, BDA, or UDA for *ddd*. For example, suppose the system disk is on a BCI-controlled drive. Edit BCIXDT.COM so that the unit number of the drive is deposited in R3. At the console-mode prompt, enter the following command:

```
>>> @BCIXDT
```

1.6.3 Booting XDELTA on a VAX 8600 or a VAX 8650 Computer

There are two ways to boot a VAX 8600 or a VAX 8650 computer with XDELTA, depending on whether the console RL02 includes a boot command file in the *dd*0XDT format, where *dd* is the device code of the system disk.

If DU0XDT is present, follow the standard boot procedure except in the following two steps:

1. When you specify the boot device, use the following command format:

   DEPOSIT R3 u

   This command deposits the unit number of the drive holding the system disk, *u*, from which to boot.

2. Then enter the following command to invoke DU0XDT:

   ```
   >>> @DU0XDT
   ```

   The command procedure boots the processor and prompts you from SYSBOOT. When the SYSBOOT> prompt appears, enter any SYSBOOT command.

To continue the booting operation, enter the CONTINUE command.

If the console media does not have the DU0XDT file, perform a normal boot procedure using an available *ddu*GEN.COM, *ddu*BOO.COM, or DEFBOO.COM procedure, including the following steps:

1. Include the /NOSTART qualifier to the BOOT command. This causes the processor to pause and prompt for console commands prior to starting the VMB initialization routines.
2. Select a value from Table 1–1 for the boot flag to control the loading of XDELTA.
3. Examine the value of the boot flag in R5. If it is not the value you want, deposit the correct value.

For example, the following commands are used to boot a VAX 8600 to include XDELTA, stop in SYSBOOT, take the initial breakpoint (flag value of 7), and continue the boot procedure:

```
>>> BOOT/NOSTART
>>> EXAMINE R5  40000000
>>> DEPOSIT R5 2  40000007
>>> CONTINUE
```

1.6.4 Booting XDELTA on a VAX 8200, 8250, 8300, or 8350 Computer

To boot a VAX 8200, 8250, 8300, or 8350 computer with XDELTA, use the following console BOOT command format:

B/R5:f devname

The boot command qualifier, /R5:f, enters a value for a flag that controls how to load XDELTA. The flag is a 32-bit hexadecimal integer loaded into R5 as input to VMB.EXE, the primary boot program. Refer to Table 1–1 for a description of the valid values for this flag. To use this qualifier, you must first modify the boot command procedure to remove (or comment out) the DEPOSIT R5 command.

The boot command procedure is specified by *devname* in the BOOT command. The *devname* format to use is *ddnu*, where *n* is the number of the VAXBI node to which the boot device unit is attached. If you do not specify *devname*, the default boot device is used.

If in R5 you specified the flag to load SYSBOOT, the SYSBOOT> prompt appears. Enter any SYSBOOT command.

For example, use the following commands to boot a VAX 8200 from the boot disk at VAXBI node 4, load XDELTA, stop in SYSBOOT, and take the initial breakpoint (that is, R5 contains 7):

```
>>> B/R5:7 DU40
SYSBOOT> CONTINUE
```

1.6.5 Booting XDELTA on a VAX 6000 Series Computer

To boot a VAX 6000 series computer with XDELTA, use the following procedure:

1. If you are booting a VAX 6000-500 computer, proceed to the next step.

   If you have a CIBCA-A adapter and are booting over the CI, insert the console tape cartridge into the console drive.

2. Press Ctrl/P to put the system in console mode.

3. Enter the BOOT command in the following format:

   BOOT /R5:a /XMI:b /BI:c [/R3:d] [/NODE:e] DUu

   For example, the following command would boot a VAX 6000 from the boot disk at VAXBI node 4 through the DWMBA adapter at XMI node E, load XDELTA, stop in SYSBOOT, and take the initial breakpoint:

   ```
   >>> BOOT /R5:7 /XMI:E /BI:4 DU1
   SYSBOOT>
   ```

   Table 1–3 lists the qualifiers to the BOOT command.

Table 1–3 Boot Command Qualifiers for the VAX 6000 Series

Qualifier	Function
/R5:a	Deposits a value (*a*) into R5. Refer to Table 1–1 for valid values for this flag.
/XMI :b	Specifies the XMI node number (*b*) of the node being accessed. Defaults to the lowest numbered I/O device.
/BI:c	Specifies the BI node number (*c*) of the node being accessed and must be used with the /XMI qualifier. Defaults to 0.
/R3:d	Specifies shadowing information. This qualifier is required only if you are using Volume Shadowing Phase I. If you are using Volume Shadowing Phase II, you need to set several SYSGEN parameters to use volume shadowing. For more information, see *Volume Shadowing for OpenVMS*.
/NODE:e	Specifies the HSC node number (*e*) of the node being accessed. The HSC node number is hexadecimal. You can specify a maximum of two HSC node numbers (if two HSCs are available). Refer to the upgrade and installation supplement for the VAX 6000 series computer.
u	Specifies the unit number of the drive holding the system disk.

1.6.6 Booting XDELTA on a VAX 7000 or VAX 10000 Series Computer

To boot a VAX 7000 or VAX 10000 series computer with XDELTA, use the following procedure:

1. Make sure the key switch is in the ENABLE position.
2. Press Ctrl/P to put the system in console mode.

3. Enter the BOOT command in the following format:

```
>>> BOOT -FLAGS g,f,e ddcu
```

BOOT can be abbreviated as B, and -FLAGS abbreviated as -F.

Table 1–4 lists the qualifiers to the BOOT command.

Table 1–4 Boot Command Qualifiers (Local and HSC Disks)

Qualifier	Function
e	Deposits a hexadecimal value (*e*) into bits 0 through 27 of R5. See Table 1–1 for valid values for this flag.
f	Deposits a hexadecimal value (*f*) into bits 28 through 31 of R5. This value affects which root will be booted.
g	Deposits a hexadecimal value (*g*) into bits 16 through 31 of R3. Required only if you are using Volume Shadowing Phase I (see *Volume Shadowing for OpenVMS* for more information). If you are using Volume Shadowing Phase II, you do not need this qualifier, but you must set several SYSGEN parameters, as described in the *Volume Shadowing for OpenVMS*.

For example, if you want to initiate a conversational boot by depositing the number 1 in bits 0 through 27 of R5 when booting the system, enter the following command:

```
>>> BOOT -FLAGS 1 DUA4
```

Using the same example, if you want to boot off root 6, enter the following command:

```
>>> BOOT -FLAGS 6,1 DUA4
```

If you also want to deposit a 40 into R3 for host-based shadowing, enter the following command:

```
>>> BOOT -FLAGS 40,6,1 DUA4
```

1.6.7 Booting XDELTA on a VAXstation 3100 or MicroVAX 3100 Series or VAXstation 4000 Series Computer

To boot XDELTA on a VAXstation 3100, MicroVAX 3100 series computer (including the VS3100-76), or a VAXstation 4000 series computer, use the following command format:

B/R5:f devname

For *devname*, enter the device name of the system disk. For *f*, enter a value from Table 1–1.

For example, to boot XDELTA and stop at the breakpoint from a drive with a device name of DKA400, enter the following command:

```
>>> B/R5:6 DKA400
```

1.6.8 Booting XDELTA on a VAXft 3000, VAXft-410, VAXft-610, or VAXft-612 Computer

To boot a VAXft 3000, VAXft-410, VAXft-610, or VAXft-612 computer with XDELTA, use the following BOOT command:

```
$ Break
>>>  BOOT/R5:7
```

This format of the BOOT command invokes XDELTA and causes SYSBOOT to prompt for input. If BREAK is not enabled, deactivate the appropriate LOCAL CONSOLE DISABLE switch on the front of the system cabinet.

1.6.9 Booting XDELTA on a MicroVAX 2000, VAXstation 2000, MicroVAX 3300/3400 Series, VAXstation 3520, VAXstation 3540, MicroVAX or VAXstation 3500/3600 Series, MicroVAX 3800/3900, VAX 4000 series, MicroVAX II, or VAX–11/750 Computer

To boot with XDELTA a MicroVAX 2000, VAXstation 2000, MicroVAX 3300/3400 series, VAXstation 3520, VAXstation 3540, MicroVAX or VAXstation 3500/3600 series, MicroVAX 3800/3900, VAX 4000 series, MicroVAX II, or VAX–11/750 computer, use the following BOOT command format:

B/n devname

The *devname* parameter is the name of the device from which to boot the system. Specify the device name using the format *ddcu*. (See the *OpenVMS System Manager's Manual* for the operating system you are using for a complete description of the format of device names.) You must specify identifiers for both the controller and the unit identifiers; there are no defaults.

The /n qualifier loads the value *n* into R5. The contents of R5 are passed as input to VMB.EXE. The value of *n* must be one of the 32-bit hexadecimal numbers described in Table 1–1.

For example, the following command boots OpenVMS on a VAX–11/750 from DUA0 with XDELTA included, stops at XDELTA's initial breakpoint, and stops in SYSBOOT to allow setting of system parameters:

```
>>> B/7 DUA0
```

The /7 qualifier includes XDELTA in the system and stops the booting process in SYSBOOT, which issues a prompt. It also stops at the breakpoint in the system-initialization routine.

You can enter SYSBOOT commands when you see the SYSBOOT> prompt.

To continue the booting operation, enter the CONTINUE command.

See the upgrade and installation supplement for your computer for more information about the B command.

1.6.10 Booting XDELTA on a VAX–11/780 or a VAX–11/785 Computer

In addition to the normal system boot command files, the console RX01 for a VAX–11/780 or VAX–11/785 computer contains the following command files that boot the system with XDELTA:

- DUAXDT.CMD
- DMAXDT.CMD

- DBAXDT.CMD

To boot the system with XDELTA, follow the procedures in the upgrade and installation supplement for your VAX computer, with the following exceptions:

- In R3, deposit the unit number of the drive holding the system disk.
- Specify one of the previous command files.

For example, if the unit number of the drive holding the system disk is 0, enter the following command:

```
>>> DEPOSIT R3 0
```

Then specify the command file that corresponds to the drive holding the system disk. For example, if the system disk is on an RA80 drive with a controller designation of A, enter the following command:

```
>>>  @DUAXDT
```

The command procedure boots the processor and prompts you from SYSBOOT. When the SYSBOOT> prompt appears, enter any SYSBOOT command.

To continue the booting operation, enter the CONTINUE command.

1.6.11 Booting a VAX–11/730 Computer Using the Console TU58

In addition to the normal system boot command files, the console TU58 for a VAX–11/730 computer contains the following command files that boot the system with XDELTA:

- DQAXDT
- DQ0XDT
- DL0XDT
- DUAXDT
- DU0XDT

To boot a VAX–11/730 computer with XDELTA, follow the procedures in the upgrade and installation supplement for your VAX computer, but specify one of the previous command files.

For example, to boot the VAX–11/730 computer from DQA1, enter the following commands:

```
>>> D/G/L 3 1
>>> @DQAXDT
```

The first command, D, deposits the unit number, 1, in R3. The second command, @DQAXDT, invokes the DQAXDT command procedure.

If the boot device is DQA0, invoke the DQ0XDT command procedure, as follows. You do not have to specify the unit number.

```
>>> @DQ0XDT
```

Either of these procedures boots the processor and prompts you from SYSBOOT. When SYSBOOT prompts you, enter any SYSBOOT command.

To continue the booting process, enter the CONTINUE command.

To boot a VAX–11/750 computer with the console TU58, refer to the upgrade and installation supplement for the VAX–11/750. The console TU58 contains the command files DUAXDT, DMAXDT, and DBAXDT, which contain the command procedures that boot the system from DU, DM, and DB devices, respectively. ♦

1.6.12 Booting XDELTA on AXP Computers

AXP

On AXP, the method for booting XDELTA is the same for all models. To boot XDELTA on any one of them, use the BOOT command as follows:

```
>>> BOOT -FLAG 0,7♦
```

1.7 Requesting an Interrupt

On AXP and VAX, if you set the boot control flag to 7, XDELTA will stop at an initial breakpoint during the system boot process. You can then set other breakpoints or examine locations in memory.

Your program can also call the routine INI$BRK, which in turn executes the first XDELTA breakpoint. Refer to Section 1.8 for the breakpoint procedure.

Once loaded into memory, XDELTA can also be invoked at any time from the console by requesting a software interrupt. For example, you might need to use a software interrupt to enter XDELTA if your program is in an infinite loop or no INI$BRK call had been made.

VAX

On VAX, INI$BRK is defined as XDELTA's breakpoint 1.

Note

Never clear breakpoint 1 from any code being debugged in XDELTA. If you accidentally clear breakpoint 1 and no other breakpoints are set, you cannot use XDELTA again until you reboot with XDELTA.♦

AXP

On AXP, INI$BRK is defined as XDELTA's breakpoint 0. It is not possible to clear breakpoint 0 from any code being debugged in XDELTA. ♦

1.7.1 Requesting Interrupts on VAX Computers

VAX

For a VAX 8530, 8550, 8600, 8650, 8810 (8700), 8820, 8820-N (8800), 8830, 8840, VAX–11/780, or VAX–11/785 computer, enter the following commands at the console terminal to request the interrupt:

```
$ Ctrl/P
>>> HALT
>>> D/I 14 E
>>> C
```

For a VAX 9000 computer, enter the following commands at the console terminal to request the interrupt:

```
$ Ctrl/P
>>> HALT/CPU=ALL
>>> D/I 14 E
>>> C/CPU=ALL
```

For a VAX 6000 series, 8200, 8250, 8300, 8350, VAX–11/730, or a VAX–11/750 computer, enter the following commands:

```
$ Ctrl/P
>>> D/I 14 E
>>> C
```

For a VAXstation 3520 or 3540 computer, perform the following steps:

1. Press and release the Halt button on the CPU control panel. When you release the Halt button, make sure it is popped out or the system will remain halted. You can also press the Break key (if enabled) on the console terminal.
2. Enter the following commands:

```
>>> D/I 14 E
>>> C/ALL
```

For a VAXft 3000, VAXft-410, VAXft-610, or VAXft-612 computer, enter the following commands at the console terminal to request the interrupt:

```
$  Break or F5
>>> HALT
>>> D/I 14 E
>>> CONT
>>> PIO
```

For a VAX 7000 or VAX 10000 series computer, enter the following commands at the console terminal to request the interrupt. If you are operating in secure mode, first set the key switch to ENABLE before entering these commands.

```
$  Ctrl/P
>>> D IPR:14 E
>>> CONT
```

For a VAXstation 2000, MicroVAX 2000, MicroVAX 3300/3400 series, MicroVAX or VAXstation 3500/3600 series, MicroVAX 3800/3900 series, VAX 4000 series, or MicroVAX II computer, perform the following steps:

1. Press and release the Halt button on the CPU control panel. When you release the Halt button, make sure it is popped out or the system will remain halted. You can also press the Break key (if enabled) on the console terminal.
2. Enter the following commands:

```
>>> D/I 14 E
>>> C
```

For an alternative method of accessing OpenVMS through a lower priority interrupt, refer to the *OpenVMS System Manager's Manual*. ♦

1.7.2 Requesting Interrupts on AXP Computers

AXP

On AXP, to request an interrupt, perform the following steps:

1. Halt the processor with the following command:

```
^P
```

2. Request an IPL 14 software interrupt with the following command:

```
>>> DEP SIRR E
```

This command deposits a 14_{10} into the software interrupt request register.

3. Reactivate the processor by issuing the CONTINUE command as follows:

```
>>> CONT
```

The process should enter XDELTA as soon as IPL drops to 14.

The following message is displayed:

```
Brk 0 at address
address! instruction
```

At this point, the exception frame is on the stack. The saved PC/PS in the exception frame tells you where you were in the program when you requested the interrupt. ♦

1.8 Accessing the Initial Breakpoint

On AXP and VAX, when debugging a program, you can set a breakpoint in the code so that XDELTA gains control of program execution.

To set a breakpoint, place a call to the system routine INI$BRK in the source code.

The INI$BRK routine contains two instructions: BPT and a second instruction that is specific to the hardware system. On systems that are not booted with XDELTA, the BPT instruction in INI$BRK is replaced with a NOP instruction.

You can use the INI$BRK routine as a debugging tool, placing calls to this routine in any part of the source code you want to debug.

VAX

On VAX, the second instruction in INI$BRK is RSB. After the break is taken, the return address (the address in the program to which control returns when you proceed from the breakpoint) is on the top of the stack.

The following command calls the INI$BRK system routine to reach the breakpoint:

```
JSB G^INI$BRK♦
```

AXP

On AXP, the second instruction in INI$BRK is JSR R31,(R26). After the break is taken, the return address (the address in the program to which control returns when you proceed from the breakpoint) is in R26.

The following C routine calls the INI$BRK system routine to reach the breakpoint:

```
extern void ini$brk(void);

main()
{
 ini$brk();
}♦
```

1.9 Proceeding from Initial XDELTA Breakpoints

VAX

On VAX, when XDELTA reaches one of its breakpoints, it displays the following message:

```
1 BRK AT nnnnnnnn

address/instruction operands♦
```

AXP

On AXP, when XDELTA reaches one of its breakpoints, it displays the following message:

```
BRK 1 AT nnnnnnnn
address!instruction operands♦
```

On AXP and VAX multiprocessor computers, the XDELTA breakpoint is taken on the processor upon which the XDELTA software interrupt was requested, which is generally the primary processor.

At this point, XDELTA is waiting for input. If you want to proceed with program execution, enter the ;P command. If you want to do step-by-step program execution, enter the S command. If you know where you have set breakpoints, examine them using the ;B command. You can also set additional breakpoints or modify existing ones.

If you entered the ;P command to proceed with program execution and the system halts with a fatal bugcheck, the system prints the bugcheck information on the console terminal. Bugcheck information consists of the following:

- Type of bugcheck
- Contents of the registers
- A dump of one or more stacks
- A list of loaded executive images

The contents of the program counter (PC) and the stack indicate where the failure was detected. Then, if the system parameter BUGREBOOT was set to 0, XDELTA issues a prompt. You can examine the system's state further by entering XDELTA commands.

1.10 Exiting from XDELTA

On AXP and VAX, XDELTA remains in memory with the operating system until you reboot without it.

2

DELTA and XDELTA Symbols and Expressions

This chapter describes how to form the symbolic expressions used as arguments to many DELTA and XDELTA commands.

2.1 Symbols Supplied by DELTA and XDELTA

DELTA and XDELTA define symbols that are useful in forming expressions and referring to registers. These symbols are described in Table 2–1.

Table 2–1 DELTA/XDELTA Symbols

Symbol	Description
.	The address of the current location. The value of this symbol is set by the Open Location and Display Contents (/), Open Location and Display Instruction (!), and the Open Location and Display Indirect (TAB) commands.
Q	The last value displayed. The value of Q is set by every command that causes DELTA or XDELTA to display the contents of memory or the value of an expression.
†Xn	Base register *n*, where *n* can range from 0 to F (hexadecimal). These registers are used for storing values, most often the base addresses of data structures in memory. For XDELTA only, XE and XF contain the addresses of two command strings that XDELTA stores in memory. See the Execute Command String (;E) command for more information. For XDELTA only, registers X4 and X5 contain specific addresses. X4 contains the address of the location that contains the PCB address of the current process on the current processor. The address that X4 contains is that of the per-CPU database for the current processor. X5 contains SCH$GL_PCBVEC, the symbolic address of the start of the PCB vector, and the list of PCB slots.
‡Xn	Base register *n*, where *n* can range from 0 to 15 (decimal). These registers are used for storing values, most often the base addresses of data structures in memory. For XDELTA only, X14 and X15 contain the addresses of two command strings that XDELTA stores in memory. See the Execute Command String (;E) command for more information. For XDELTA only, registers X4 and X5 contain specific addresses. X4 contains the address of the location that contains the PCB address of the current process on the current processor. The address that X4 contains is that of the per-CPU database for the current processor. X5 contains SCH$GL_PCBVEC, the symbolic address of the start of the PCB vector, and the list of PCB slots.

†VAX specific
‡AXP specific

(continued on next page)

Table 2–1 (Cont.) DELTA/XDELTA Symbols

Symbol	Description
†Rn	General register *n*, where *n* can range from 0 to F (hexadecimal). RF+4 is the processor status longword (PSL), RE is the stack pointer, and RF is the program counter (PC).
‡Rn	General register *n*, where *n* can range from 0 to 31 (decimal). PS is the processor status and PC is the program counter.
†Pn	The internal processor register at processor address *n*, where *n* can range from 0 to 3F (hexadecimal). See the *VAX Architecture Reference Manual* for a description of these processor registers.
‡P(IPR name)	The internal processor register at processor address *IPR name*. See the *Alpha Architecture Reference Manual* for the names and descriptions of these processor registers.
‡FPCR	The floating point control register.
†G	^X80000000, the prefix for system space addresses. G2E, for example, is equivalent to ^X8000002E.
‡G	^XFFFFFFFF80000000, the prefix for system space addresses.
H	^X7FFE0000, the prefix for addresses in the control region (P1 space). H2E, for example, is equivalent to ^X7FFE002E.
‡PID:Rn	The internal PID of another process that you want to look at. Rn represents the register that you want to read or write. Rn applies to R0-R29, PC, and PS. An address or address range can be specified instead of Rn for the / command, as described in Chapter 4.

†VAX specific
‡AXP specific

AXP

On AXP, note that the floating point registers cannot be accessed by DELTA or XDELTA. ♦

2.2 Forming Numeric Expressions

Expressions are combinations of numbers, symbols that have numeric values, and arithmetic operators.

VAX

On VAX, both DELTA and XDELTA store and display all numbers in hexadecimal. They also interpret all numbers as hexadecimal.♦

AXP

On AXP, all numbers except integer and floating point registers are stored and displayed in hexadecimal. These registers are stored and displayed in decimal.♦

On AXP and VAX, expressions are formed using regular (infix) notation. Both DELTA and XDELTA ignore operators that trail the expression. The following is a typical expression (in hexadecimal):

```
G4A32+24
```

DELTA and XDELTA evaluate expressions from left to right. No operator takes precedence over any other.

DELTA and XDELTA recognize five binary arithmetic operators, one of which also acts as a unary operator. They are listed in Table 2–2.

Table 2–2 Arithmetic Operators

Operator	Action
+ or SPACE	Addition
-	Subtraction when used as a binary operator, or negation when used as a unary operator
*	Multiplication
%	Division
@	Arithmetic shift

The following example shows the arguments required by the arithmetic-shift operator:

```
n@j
```

In this example, *n* is the number to be shifted, and *j* is the number of bits to shift it. If *j* is positive, *n* is shifted to the left; if *j* is negative, *n* is shifted to the right. Argument *j* must be less than 20_{16} and greater than -20_{16}. Bits shifted beyond the limit of the longword are lost; therefore, the result must fit into a longword.

3 Debugging Programs

When you use DELTA or XDELTA, there are no prompts, few symbols, and one error message. You move through program code by referring directly to address locations. This chapter provides directions for the following actions:

- Referencing addresses
- Referencing registers, the PSL or PS, and the stack
- Interpreting the error message
- Debugging kernel mode code under certain conditions
- Debugging an installed, protected, shareable image
- Using XDELTA on multiprocessor computers

AXP

- Debugging code when single-stepping fails (AXP only)
- Debugging code that does not match the compiler listings (AXP only)♦

For examples of DELTA debugging sessions on VAX and AXP, refer respectively to Appendix A and Appendix B.

3.1 Referencing Addresses

When using DELTA or XDELTA to debug programs, you move through the code by referring to addresses. To help you identify address locations within your program, use a list file and a map file. The list file (.LIS) lists each instruction and its offset value from the base address of the program section. The full map file (.MAP) lists the base addresses for each section of your program. To determine the base address of a device driver program, refer to the *OpenVMS VAX Device Support Manual*.

Once you have the base addresses of the program sections, locate the instruction in the list file where you want to start the debugging work. Add the offset from the list program to the base address from the map file. Remember that all calculations of address locations are done in hexadecimal. You can use DELTA/XDELTA to do the calculations for you with the = command.

To make address referencing easier, you can use offsets to a base address. Then you do not have to calculate all address locations. First, place the base address into a base register. Then move to a location using the offset to the base address stored in the register.

Whenever DELTA/XDELTA displays an address, it will display a relative address if the offset falls within the permitted range (see the ;X command in Chapter 4).

3.1.1 Referencing Addresses (VAX Only)

VAX

On VAX, to reference addresses during a DELTA debug session, use the following example as a guide. The example uses a simple VAX MACRO program (EXAMPLE.MAR). You can also use the same commands in an XDELTA debugging session.

```
0000    1   .title  example
0000    2
0000    3   .entry  start   ^M<r3,r4>
0002    4           clrl    r3
0004    5           movl    #5,r4
0007    6   10$:    addl    r4,r3
000A    7           sobgtr  r4,10$
000D    8           ret
000E    9
000E   10   .end    start
```

The following procedure generates information to assist you with address referencing:

1. Use the /LIST qualifier to assemble the program and generate the list file.

   To generate the list file for the previous example, use the following command:

   ```
   $ MACRO/LIST EXAMPLE
   ```

2. Use the /MAP qualifier with the link command to generate the full map file (.MAP file). Make sure that the default /DEBUG or /TRACEBACK qualifier is active for your link command. If not, specify /DEBUG or /TRACEBACK along with the /MAP qualifier.

   To generate the map file for the example program, use the following command:

   ```
   $ LINK/MAP EXAMPLE
   ```

3. Refer to the Program Section Synopsis of the map file, locate the section that you want to debug, and look up the base address.

   For the example program, the map file is EXAMPLE.MAP. A portion of the Program Section Synopsis is shown below. The first section of the program has a base address of 200.

   ```
                    +--------------------------+
                    ! Program Section Synopsis !
                    +--------------------------+

   Psect Name      Module Name   Base     End          Length
   ----------      -----------   ----     ---          ------

   . BLANK .                     00000200 0000020D 0000000E (         14.)
                   EXAMPLE       00000200 0000020D 0000000E (         14.)
   ```

4. Refer to the list file for the location of the specific instruction where you want to start debugging.

   For the example program, start with the second instruction (MOVL #5,R4) with an offset of 4.

5. Enable DELTA using the following commands:

   ```
   $ DEFINE LIB$DEBUG SYS$LIBRARY:DELTA
   $ RUN/DEBUG EXAMPLE
   ```

6. If you want to store the base address in a base register, use the ;X command to load the base register.

For the example program, use the following DELTA/XDELTA command to store the base address 200 in base register 0.

```
200,0;X Return
```

7. Now you can move to specific address locations.

   For example, if you want to place a breakpoint at the second instruction (MOVL #5,R4), you would calculate the address as 200 (base address) plus 4 (offset), or 204, and specify the ;B command as follows:

   ```
   204;B Return
   ```

   Alternatively, if you stored the base address in the base register, you could use the address expression X0+4 (or "X0 4", where the + sign is implied), as follows:

   ```
   X0+4;B Return
   ```

Reverse this technique to find an instruction displayed by DELTA/XDELTA in the .LIS file, as follows:

1. Note the address of the instruction you want to locate in the .LIS file.

   For example, DELTA/XDELTA displays the following instruction at address 020A:

   ```
   20A!  sobgtr  r4,00000207
   ```

   The following steps allow you to find the instruction at location 207:

2. Refer to the .MAP file and identify the PSECT and MODULE where the address of the instruction is located. Check the base address value and the end address value of each PSECT and MODULE. When the instruction address is between the base and end address values, record the PSECT and MODULE names.

   In the example, the instruction address is located in the EXAMPLE module (.BLANK. psect). The address instruction, 207, is between the base address 200 and the end address 20D.

3. Subtract the base address from the instruction address. Remember that all calculations are in hexadecimal and that you can use the DELTA/XDELTA = command to do the calculations. The result is the offset.

   For the example, subtract the base address 200 from the instruction address 207. The offset is 7.

4. Refer to the .LIS file. Look up the MODULE and then find the correct PSECT. Look for the offset value you calculated in the previous step.

   In the example, there is only one PSECT and MODULE. Look up the instruction at offset 7. The program is branching to the following instruction:

   ```
   10$:   addl r4,r3♦
   ```

3.1.2 Referencing Addresses (AXP Only)

AXP

On AXP, to reference addresses during a DELTA debug session, use the following example as a guide. The example uses a simple C program (HELLO.C). You can also use the same commands in an XDELTA debug session.

```
#include <stdio.h>

main()
{
 printf("Hello world\n");
}
```

The following procedure generates information to assist you with the address referencing:

1. Use the /LIST and /MACHINE_CODE qualifiers to compile the program and generate the list file containing the Alpha machine instructions.

   To generate the list file for the previous example, use the following command:

   ```
   $ cc/list/machine_code hello
   ```

   The compiler will generate the following Alpha code in the machine code portion of the listing file:

   ```
      .PSECT $CODE, OCTA, PIC, CON, REL, LCL, SHR,-
       EXE, NORD, NOWRT
   0000    main::                                                  ; 000335
   0000            LDA     SP, -32(SP)             ; SP, -32(SP)
   0004            LDA     R16, 48(R27)            ; R16, 48(R27)      ; 000337
   0008            STQ     R27, (SP)               ; R27, (SP)         ; 000335
   000C            MOV     1, R25                  ; 1, R25            ; 000337
   0010            STQ     R26, 8(SP)              ; R26, 8(SP)        ; 000335
   0014            STQ     FP, 16(SP)              ; FP, 16(SP)
   0018            LDQ     R26, 32(R27)            ; R26, 32(R27)      ; 000337
   001C            MOV     SP, FP                  ; SP, FP            ; 000335
   0020            LDQ     R27, 40(R27)            ; R27, 40(R27)      ; 000337
   0024            JSR     R26, DECC$GPRINTF       ; R26, R26
   0028            MOV     FP, SP                  ; FP, SP            ; 000338
   002C            LDQ     R28, 8(FP)              ; R28, 8(FP)
   0030            LDQ     FP, 16(FP)              ; FP, 16(FP)
   0034            MOV     1, R0                   ; 1, R0
   0038            LDA     SP, 32(SP)              ; SP, 32(SP)
   003C            RET     R28                     ; R28
   ```

   Notice the statement numbers on the far right of some of the lines. These numbers correspond to the source line statement numbers from the listing file as shown next:

   ```
   335 main()
   336 {
   337     printf("Hello world\n");
   338 }
   ```

2. Use the /MAP qualifier with the link command to generate the full map file (.MAP file). To produce a debuggable image, make sure that either /DEBUG or /TRACEBACK (the default) is also specified with the link command.

   To generate the map file for the example program, use the following command:

   ```
   $ LINK/MAP/FULL HELLO
   ```

3. Refer to the Program Section Synopsis of the map file. Locate the code section that you want to debug and its base address.

For the example program, the map file is HELLO.MAP. A portion of the Program Section Synopsis is shown below. The $CODE section of the program has a base address of 20000.

```
  +--------------------------+
  ! Program Section Synopsis !
  +--------------------------+

Psect Name      Module Name       Base     End          Length
----------      -----------       ----     ---          ------
$LINKAGE                       00010000 0001007F 00000080 (       128.)
                HELLO          00010000 0001007F 00000080 (       128.)
$CODE                          00020000 000200BB 000000BC (       188.)
                HELLO          00020000 000200BB 000000BC (       188.)
```

4. Refer to the list file for the location where you want to start debugging. First find the source line statement number. Next find that statement number in the machine code listing portion of the list file. This is the specific instruction where you want to start debugging.

   For the example program, source statement 337 is the following:

   ```
   printf("Hello world\n");
   ```

   Search the machine code listing for statement 337. The first occurrence is the instruction at offset 4 from the start of "main::" and the base of the $CODE PSECT.

5. Enable DELTA using the following commands:

   ```
   $ DEFINE LIB$DEBUG SYS$LIBRARY:DELTA
   $ RUN/DEBUG HELLO
   ```

6. If you want to store the base address in a base register, use the ;X command to load the base register.

   For the example program, use the following DELTA/XDELTA command to store the base address of 20000 in base register 0.

   ```
   20000,0;X
   ```

7. Now you can move to specific address locations.

   For example, if you want to place a breakpoint at offset 4, you would calculate the address as 20000 (base address) plus 4 (offset), or 20004, and specify the ;B command as follows:

   ```
   20004;B
   ```

   Alternatively, if you stored the base address in the base register, you could use the address expression X0+4 (or "X0 4", where the + sign is implied) to set the breakpoint as follows:

   ```
   X0+4;B
   ```

Reverse this technique to find an instruction displayed by DELTA/XDELTA in the .LIS file, as follows:

1. Note the address of the instruction you want to locate in the .LIS file.

   For example, DELTA/XDELTA displays the following instruction at address 20020:

   ```
   20020!  LDQ             R27,#X0028(R27)
   ```

   The following steps allow you to find this instruction in the .LIS file.

2. Refer to the .MAP file, and identify the psect and module where the address of the instruction is located. Check the base address value and the end address value of each psect and module. When the instruction address is between the base and end address values, record the psect and module names.

   In the example, the instruction address is located in the HELLO module ($CODE PSECT). The address, 20020, is between the base address 20000 and the end address 200BB.

3. Subtract the base address from the instruction address. Remember that all calculations are in hexadecimal and that you can use the DELTA/XDELTA = command to do the calculations. The result is the offset.

   For example, subtract the base address of 20000 from the instruction address 20020. The offset is 20.

4. Refer to the .LIS file. Look up the module and then find the correct psect. Look for the offset value you calculated in the previous step.

   In the example, there are two psects and one module but only one $CODE psect. Look up the instruction at offset 20, and you will find the following in the .LIS file:

```
0020        LDQ R27, 40(R27)            ; R27, 40(R27)      ; 000337♦
```

3.2 Referencing Registers (VAX Only)

VAX

On VAX, to view the contents of the 16 general registers (including the program counter and the stack pointer) and the processor status longword (PSL), use the same DELTA/XDELTA commands as you use to view the contents of any memory location (for example, the /, LINEFEED, and the ESC commands). The symbols used to identify the locations of the registers and PSL are as follows:

- The general registers are referred to by the symbol R and a hexadecimal number from 0_{16} to F_{16} representing the number of the register. For example, general register 1_{10} is $R1_{16}$ and general register 10_{10} is RA_{16}. The stack pointer is located in general register 14_{10}, RE_{16}. The program counter is in general register 15_{10}, RF_{16}.
- Upon entry to DELTA or XDELTA, the PSL is stored in the longword directly following the longword representing general register F_{16}. Reference it by using the general register F_{16} symbol plus a longword (RF+4). ♦

3.3 Referencing Registers (AXP Only)

AXP

On AXP, to view the contents of the 32 integer registers, including the program counter (PC), the stack pointer (SP), the processor status (PS), the 32 floating point registers, the floating point control register (FPCR), and internal processor registers (IPRs), use the same DELTA/XDELTA commands that you use to view the contents of any memory location. These commands include /, LINEFEED, and ESC. The symbols for identifying these registers follow:

- Integer registers are referenced by the symbol R and a decimal number from 0 to 31. For example, register 1_{10} is $R1_{10}$ and register 10_{10} is $R10_{10}$. (Decimal notation differs from the original implementation on VAX which uses hexadecimal notation.)
- PC is referenced symbolically by PC.
- PS is referenced symbolically by PS.

- FP is referenced by R29.
- SP is referenced by R30.
- Floating point registers are referenced by FP and a decimal number from 0 to 31. For example, floating point register 1_{10} is $FP1_{10}$ and floating point register 10_{10} is $FP10_{10}$.
- FPCR is treated like any other floating point register except, to explicitly open it, you specify FPCR/.
- Internal processor registers (IPRs) are accessed symbolically, for example, P(ASTEN). For IPR names, see the *Alpha Architecture Reference Manual.* ♦

3.4 Interpreting the Error Message

On AXP and VAX, when you make an error entering a command in DELTA or XDELTA, you get the Eh? error message. This is the only error message generated by DELTA or XDELTA. It is displayed if you enter an invalid command or reference an address that cannot be displayed.

AXP

On AXP, the error message Eh? is also displayed if you are unable to single-step or proceed due to no write access to next location. ♦

3.5 Debugging Kernel Mode Code Under Certain Conditions

On AXP and VAX, some programs exist which, while running in process space, change mode to kernel and raise IPL. Typically, this code is debugged with both DELTA and XDELTA. DELTA is used to debug the kernel mode code at IPL zero. XDELTA is used to debug the code at elevated IPL. (DELTA does not work at elevated IPL.)

Before you can debug such code with XDELTA on an AXP or VAX computer, you must do some setup work.

3.5.1 Setup Required (VAX Only)

VAX

On VAX, some setup work is required before you can debug kernel mode code that runs in process space at an elevated IPL. Before you access XDELTA, do the following:

1. Ensure that page faults do not occur at elevated IPL by locking into memory (or the working set) the code that runs at elevated IPL.
2. Make the code writable if you plan to do anything more than single-step through your code (such as set breakpoints, step-overs, and so forth). (By default, code pages are read only.) To make the code writable, modify the code psect attributes in the link options file or set the affected code pages to writable with $SETPRT. ♦

3.5.2 Setup Required (AXP Only)

AXP

On AXP, some setup work is required before you can debug kernel mode code that runs in process space at an elevated IPL. Before you access XDelta, do the following:

1. Ensure that page faults do not occur at elevated IPL by locking into memory (or the working set) the code that runs at elevated IPL.

2. Make the code writable. (By default, code pages are read only.) To do this, modify the code psect attributes in the link options file or set the affected code pages to writable with $SETPRT.
3. Make code pages copy-on-reference (CRF). You can do this when you make the code writable. If you modify the link options file, set the code psect attributes to be WRT, NOSHR. If you use $SETPRT, it automatically makes the pages CRF.♦

3.5.3 Accessing XDELTA

On AXP and VAX, after you set up the code for debugging, you are ready to access XDELTA. The most convenient method is to invoke INI$BRK from the code at elevated IPL. This causes a trap into XDELTA. You can then step out of the INI$BRK routine into the code to be debugged.

3.6 Debugging an Installed, Protected, Shareable Image

Some shareable images, such as user-written system services, must be linked and installed in a way that precludes debugging with DELTA unless you take further steps. Those steps are described in this section.

Typically, a user-written system service is linked and installed in such a way that the code is shared in a read-only global section, the data is copy-on-reference, and the default code psects are read-only and shareable. Such a shareable image is installed with the Install utility using a command like the following:

```
INSTALL> myimage.exe /share/protect/open/header
```

Other qualifiers can also be used.

When installed in this way, the shareable image code is read-only. However, to debug a user-written system service with DELTA, to single-step and to set breakpoints, the code must either be writable or DELTA must be able to change the code page protection to make it writable. Neither is possible when the code resides in a read-only global section.

Therefore, to debug a user-written system service, you must link and install it differently. In linking the image, the code psects must be set to writable and, preferably, to nonshareable (to force the code pages to be copy-on-reference). Multiple processes accessing this code through the global section will each have their own private copy. You can do this in the link options file by adding a line such as the following for each code psect:

```
PSECT=$CODE$,NOSHR,WRT
```

Then, the image must be installed writable with the /WRITE qualifer and without the /RESIDENT qualifier, as follows:

```
INSTALL> myimage.exe /share/protect/open/header/write
```

After you have installed the image in this way, you can use DELTA to set breakpoints in the shareable image code and single-step through it.

3.7 Using XDELTA on Multiprocessor Computers

On AXP and VAX multiprocessor computers, only one processor can use XDELTA at a time. If a second processor attempts to enter XDELTA when another processor has already entered it, the second processor waits until the first processor has exited XDELTA. If the processor using XDELTA sets a breakpoint, other processors are aware of the breakpoint. Therefore, when the code with the XDELTA breakpoint is executed on another processor, that processor will enter XDELTA and stop at the specified breakpoint.

XDELTA uses its own system control block (SCB) to direct all interrupt handling to an error handling routine in XDELTA. Therefore, an error encountered by XDELTA does not affect any other processors that share the standard system SCB.

VAX

On VAX, when a breakpoint is taken by a processor in a multiprocessor environment, the processor's physical identification number is displayed on the XDELTA breakpoint message line as a 2-digit hexadecimal number. The following is an example of a breakpoint message in a multiprocessor environment:

```
1 BRK AT 00000400 ON CPU 03
00000400/movl #5,r4♦
```

AXP

On AXP, the processor's physical identification number is similarly displayed but the number is decimal instead of hexadecimal with no leading zeros. For example:

```
BRK 1 AT 20000 ON CPU 2
20000! LDL R1,(R2)♦
```

3.8 Debugging Code When Single-Stepping Fails (AXP Only)

AXP

On AXP, the use of the S command to single-step occasionally fails and the error message Eh? is displayed. This can happen either when you are single-stepping through code or when you have stopped at a breakpoint. In each case, it fails because XDELTA does not have write access to the next instruction. Directions on how to continue debugging for both cases follow:

- You are single-stepping through your code and your single-step fails.

  You can set other breakpoints and proceed with the ;P command. If this occurs at a JSR or BSR instruction, you can first use the O command and then either single-step (with the S command) or proceed (with the ;P command).

- You have stopped at a breakpoint and your attempt to single-step fails.

  You can delete the breakpoint and then proceed with the ;P command. If this occurs at a JSR or BSR instruction, it may be possible to first use the O command and then either single-step (with the S command) or proceed (with the ;P command).

3.9 Debugging Code That Does Not Match the Compiler Listings (AXP Only)

On AXP, there are two cases when the code in your image does not exactly match your compiler listings. As long as you understand why these differences exist, they should not interfere with your debugging. The explanations follow:

- The compilers generate listings with mnemonics that replace some of the AXP assembly language instructions. This makes the listings easier to read but can initially cause confusion because the code does not exactly match the code in your image. In every case, there is a 1–to–1 correlation between the line of code in your image and the line of code in your listing.
- In certain situations, the linker can modify the instructions in your image so that they do not exactly match your compiler listings. Typically, the linker is replacing JSR instructions and the call setup to use a BSR instruction for better performance.♦

4

DELTA/XDELTA Commands

This chapter describes how to use each DELTA and XDELTA command to debug a program. It also describes which commands are used only with DELTA. Table 4–1 provides a summary of the DELTA/XDELTA commands that are common to OpenVMS VAX and OpenVMS AXP. Table 4–2 provides a summary of the DELTA/XDELTA commands that are available only on OpenVMS AXP.

Each command in this chapter includes an example. The program used for all the examples, except those illustrating commands available only on OpenVMS AXP, is the program listed in Appendix A.

4.1 Command Usage Summary

DELTA and XDELTA use the same commands with the following exceptions:

- Only DELTA uses the EXIT and ;M commands and arguments that specify a process identification.
- XDELTA defines some base registers that DELTA does not (refer to Chapter 2).
- AXP On AXP, only DELTA uses the ;I command. ♦

For AXP and VAX, all differences are noted in command descriptions.

Enter the LINEFEED, ESC, TAB, and RETURN commands by pressing the corresponding key.

Table 4–1 DELTA/XDELTA Command Summary

Command	Description
[	Set Display Mode
/	Open Location and Display Contents in Prevailing Width Mode
!	Open Location and Display Contents in Instruction Mode
LINEFEED	Close Current Location, Open Next
ESC	Open Location and Display Previous Location
TAB	Open Location and Display Indirect Location
"	Open Location and Display Contents in ASCII Mode
RETURN	Close Current Location
;B	Breakpoint
;P	Proceed from Breakpoint
;G	Go

(continued on next page)

Table 4–1 (Cont.) DELTA/XDELTA Command Summary

Command	Description
S	Step Instruction
O	Step Instruction over Subroutine
'string'	Deposit ASCII String
;E	Execute Command String
;X	Load Base Register
=	Display Value of Expression
†;M	Set All Processes Writable (available only on DELTA)
‡;M	Set All Processes Writable; also, set selected registers of other processes writable (available only on DELTA)
;L	Lists Names and Locations of Loaded Executive Images
EXIT	Exit from DELTA debugging session

†VAX specific
‡AXP specific

AXP

The commands in Table 4–2 are available only on OpenVMS AXP. ♦

Table 4–2 DELTA/XDELTA Command Summary (AXP Only)

Command	Description
;Q	Validate queue
;C	Force system to bugcheck and crash
;W	Locate and display the executive image that contains the specified address
;I	Locate and display information about the current main image that contains the specified address; also display information about all shareable images activated by the current main image (available only on DELTA)
;H	Display on video terminal or at hardcopy terminal
\string\	Display the ASCII text string enclosed in backslashes

[(Set Display Mode)

Sets the width mode of displays produced by DELTA/XDELTA commands.

Format

```
[ mode
```

Argument

mode
Specifies the display mode as follows:

Mode	Meaning
B	Byte mode. Subsequent open and display location commands display the contents of one byte of memory.
L	Longword mode. Subsequent open and display location commands display the contents of a longword of memory. This is the default mode.
W	Word mode. Subsequent open and display location commands display the contents of one word of memory.

AXP

On AXP, the following modes are also available.

Mode	Meaning
A	Address display of 32-bit/64-bit. Subsequent address displays will be 64 bits.
Q	Quadword mode. Subsequent open and display location commands display the contents of a quadword of memory.♦

Description

The Set Display Mode command changes the prevailing display width to byte, word, longword, or quadword. The default display width is longword. The display mode remains in effect until you enter another Set Display Mode command.

Example

```
R0/   00000001    ❶
[B                ❷
R0/   01          ❸
```

❶ Contents of general register 0 (R0) are displayed using the / command. The display is the default mode, longword.

❷ Display mode is changed to byte mode using the [B command.

❸ Contents of R0 are displayed in byte mode. The least significant byte is displayed.

/ (Open Location and Display Contents in Prevailing Width Mode)

Opens a location and displays its contents in the prevailing display mode.

Format

[pid:][start-addr-exp][,end-addr-exp]/ current-contents [new-exp]

Arguments

pid
The internal process identification (PID) of a process you want to access. If you specify zero or do not specify a PID, the default process is the current process. This argument cannot be used with XDELTA.

If you use the **pid** argument, every time you use this command during the debugging session the contents of the same process are displayed, unless you specify a different **pid**.

You can obtain the internal PID of processes by running the System Dump Analyzer utility (SDA). Use the SDA command SHOW SUMMARY to determine the external PID. Then use the SDA command SHOW PROCESS/INDEX to determine the internal PID. Refer to your operating system's *System Dump Analyzer Utility Manual* for more information about using SDA commands.

Note

The register examples in the descriptions of **start-addr-exp** and **end-addr-exp** apply to both VAX and AXP registers. (AXP register numbers are displayed in decimal, and VAX register numbers are displayed in hexadecimal.)

start-addr-exp
The address of the location to be opened, or the start of a range of addresses to be opened. If not specified, the address displayed is that currently specified by the symbol Q (last quantity displayed). Use the following syntax to display a single address location:

start-addr-exp/

You can also specify a register for this parameter. For example, if you want to view the contents of general register 3 (R3), enter the following DELTA/XDELTA command:

```
R3/
```

end-addr-exp
The address of the last location to be opened. Use the following syntax to display a range of address locations:

start-addr-exp,end-addr-exp/

You can also specify a range of registers. For example, if you want to view the contents of general registers 3 through 5, enter the following DELTA/XDELTA command:

```
R3,R5/
```

If you specify an address expression for **end-addr-exp** that is less than **start-addr-exp**, DELTA/XDELTA displays the contents of **start-addr-exp** only.

current-contents
You do not specify this parameter. It is a hexadecimal value, displayed by DELTA /XDELTA, of the contents of the location (or range of locations) you specified with the **pid** argument and the address expression. It is displayed in the prevailing width display mode.

new-exp
An expression, the value of which is deposited into the location just displayed. If you specify **new-exp** after a range of locations, the new value is placed only in the last location (specified by **end-addr-exp**).

When you specify **new-exp**, terminate the command by pressing the Return key.

If you want to deposit a new value into a location in another process (that is, you specified a PID other than the current process), you must have already set the target process to be writable using the ;M command.

If the value you deposit is longer than the last location where it will be deposited, the new value overwrites subsequent locations. For example, the values at address locations 202 and 204 are as follows:

```
202/ 05D053D4
204/ C05405D0
```

If you deposited the value FFFFFFFFF at address 202, the overflow value would overwrite the value stored at address location 204, as follows:

```
202/ 05D053D4  FFFFFFFFF Return
204/ C054FFFF
```

Description

The Open Location and Display Contents command opens the location or range of locations at **start-addr-exp** and displays **current-contents**, the contents of that location, in hexadecimal format. You can place a new value in the location by specifying an expression. A new value overwrites the last value displayed.

To display a range of locations, give the **start-addr-exp** argument as the first address in the range, followed by a comma, followed by the last address in the range (the **end-addr-exp** argument). For example, if you want to display all locations from 402 to 4F0, the command is as follows:

```
402,4F0/
```

This command changes the current address (. symbol) to the contents of the opened location. A subsequent Close Location command does not change the current address. However, a subsequent Close Current Location and Open Next command (ESC or LINEFEED) executes as follows:

- Writes any **new-exp** specified
- Closes the location opened by the / command
- Adds the number of bytes (defined by the prevailing display width mode) to the address just opened with the / command
- Changes the current address to the new value
- Opens the new location and displays the contents

The display mode remains hexadecimal until the next Open Location and Display Contents in Instruction Mode (!) command or Open Location and Display Contents in ASCII Mode (") command.

In DELTA, not XDELTA, processes having the CMKRNL privilege can examine the address space of any existing process. Use **pid** to specify the internal PID of the process you want to examine. For example, use the following command to view address location 402 in the process with a PID of 00010010:

```
00010010:402/
```

AXP

On AXP, DELTA also permits the examination of general purpose registers in another process. The PID specifies the internal PID of the process you want to examine. For example, use the following command to examine R5 in the process with a PID of 00010010:

```
00010010:R5/ ♦
```

Example

```
R0,R9/00000001
R1/00000000
R2/00000226
R3/7FF2AD94
R4/000019B4
R5/00000000
R6/7FF2AA49
R7/8001E4DD
R8/7FFED052
R9/7FFED25A
```

Contents of all the general registers R0 through R9 are displayed.

! (Open Location and Display Contents in Instruction Mode)

Displays an instruction and its operands.

Format

[pid:][start-addr-exp][,end-addr-exp] !

Arguments

pid
The internal process identification (PID) of a process you want to access. If you specify zero, or do not specify any PID, the default process is the current process. This argument cannot be used with XDELTA.

Subsequent open location and display contents commands, issued after using the **pid** argument, display the contents of the location of the specified process until you specify another PID with this command.

You can obtain the internal PID of processes by running the System Dump Analyzer utility (SDA). Use the SDA command SHOW SUMMARY to determine the external PID. Then use the SDA command SHOW PROCESS/INDEX to determine the internal PID. Refer to your operating system's *System Dump Analyzer Utility Manual* for more information about SDA commands.

start-addr-exp
The address of the instruction, or the first address of the range of instructions, to display. If you do not specify this parameter, the address displayed is that currently specified by Q (last quantity displayed). When you want to view just one location, the syntax is as follows:

start-addr-exp!

end-addr-exp
The address of the last instruction in the range to display. When you want to view several instructions, the syntax is as follows:

start-addr-exp,end-addr-exp!

Each location within the range is displayed with the address, a slash (/), and the MACRO instruction.

Description

The Open Location and Display Contents in Instruction Mode command displays the contents of a location or range of locations as a MACRO instruction. DELTA /XDELTA does not make any distinction between reasonable and unreasonable instructions or instruction streams.

This command does not allow you to modify the contents of the location. The command sets a flag that causes subsequent Close Current Location and Display Next (LINEFEED) and Open Location and Display Indirect Location (TAB) commands to display MACRO instructions. You can clear the flag by using the Open Location and Display Contents (/) command, which displays the contents of the location as a hexadecimal number, or Open Location and Display Contents in ASCII Mode ("), which displays the contents of the location in ASCII.

! (Open Location and Display Contents in Instruction Mode)

When an address appears as an instruction's operand, DELTA/XDELTA sets the Q symbol to that address. Then enter ! again to go to the address specified in the instruction operand. DELTA/XDELTA changes Q only for operands that use program-counter or branch-displacement addressing modes; Q is not altered for operands that use literal and register addressing modes. This feature is useful for following branches.

Example

VAX

The following example applies only to OpenVMS VAX.

```
69B!BRB    0000067A ❶
!CLRQ      -(SP)    ❷
```

❶ The instruction at address 69B is displayed using the ! command. DELTA /XDELTA displays a branch instruction and sets Q (last address location displayed) to the branch address 67A.

❷ The instruction at address 67A is displayed using the ! command. The value of Q is used as the address location.♦

Example

AXP

The following example applies only to OpenVMS AXP.

```
30000!       LDA        SP,#XFFE0(SP) ❶
00030004!     BIS        R31,R31,R18 ❷
```

❶ The instruction at address 30000 is displayed using the ! command. DELTA /XDELTA displays a LDA instruction. Note that unlike on a VAX computer, an absolute address never appears in an instruction operand. So the value of Q has no use after an instruction display.

❷ After typing a LINEFEED command, DELTA/XDELTA displays the next instruction location and the instruction at that address.♦

LINEFEED (Close Current Location, Open Next)

Closes the currently open location and opens the next location, displaying its contents.

Format

LINEFEED

Description

The Close Current Location Open Next command closes the currently open location, then opens the next and displays its contents. This command accepts no arguments, and thus can only be used to open the next location. It is useful for examining a series of locations one after another. First, set the location where you want to start (for example, with the / or (!) command). Then, press the Linefeed key repeatedly to examine each successive location.

The LINEFEED command displays the contents of the next location in the prevailing display mode and display width. If the current display mode is hexadecimal (the / command was used) and the display width is word, the next location displayed is calculated by adding a word to the current location. Its contents are displayed in hexadecimal. If the current display mode is instruction, the next location displayed is the next instruction, and the contents are displayed as a MACRO instruction.

On keyboards without a separate Linefeed key, press Ctrl/J. The Linefeed key on LK201 keyboards (VT220, VT240, VT340, and workstation keyboards) generates different characters and cannot be used for the LINEFEED command. You must use Ctrl/J.

This command is useful for displaying a series of MACRO instructions, a series of register values, or a series of values on the stack.

The values in the symbol Q and the symbol . are changed automatically.

Example

VAX

The following example applies only to OpenVMS VAX.

```
6B9!CLRQ          -(SP) ❶ [Linefeed] ❷
000006BB/CLRQ     -(SP)   [Linefeed]
000006BD/PUSHL    X1+002E [Linefeed]
000006C1/PUSHAL X1+003A   [Linefeed]
000006C5/CLRQ     -(SP)   [Linefeed]
000006C7/PUSHL    #00
```

❶ Instruction at address 6B9 is displayed using the ! command.

❷ Five successive instructions are displayed by pressing the Linefeed key five times. The LINEFEED command is not echoed on the terminal.♦

Example

AXP

The following example applies only to OpenVMS AXP.

```
30000!          LDA             SP,#XFFE0(SP)  ❶
00030004!       BIS             R31,R31,R18    ❷
00030008!       STQ             R27,(SP)
0003000C!       BIS             R31,R31,R19
00030010!       STQ             R26,#X0008(SP)
00030014!       BIS             R31,#X04,R25
```

❶ Instruction at address 30000 is displayed using the ! command.

❷ Five successive instructions are displayed by pressing the Linefeed key five times. The LINEFEED command is not echoed on the terminal.♦

ESC (Open Location and Display Previous Location)

Opens the previous location and displays its contents.

Format

ESC

Description

The Open Location and Display Previous Location command decrements the location counter (.) by the width (in bytes) of the prevailing display mode, opens that many bytes, and displays the contents on a new line. The address of the location is displayed on the new line in the prevailing mode, followed by a slash (/) and the contents of that address.

VAX

On VAX, this command is ignored if the prevailing display mode is instruction mode (set by the ! command).♦

On AXP and VAX, use this command to move backwards through a series of locations. Set the address where you want to start (for example, with the / command). Then press the ESC key repeatedly to display each preceding location. ESC is echoed as a dollar sign ($) on the terminal.

On keyboards without a separate ESC key, press Ctrl/3 or the escape key sequence that you defined on your keyboard. The ESC key on LK201 keyboards (VT220, VT240, VT340, and workstation keyboards) generates different characters and cannot be used for the ESC command. You must use Ctrl/3.

Example

```
R1/00000000 ❶ $ ❷ ESC
R0/00000001
```

❶ The contents of general register 1 are displayed using the / command.

❷ The contents of general register 0, the location prior to general register 1, are displayed by pressing ESC.

TAB (Open Location and Display Indirect Location)

Opens the location addressed by the contents of the current location and displays its contents.

Format

TAB

Description

The Open Location and Display Indirect Location command opens the location addressed by the contents of the current location and displays the contents of the addressed location on a new line. The display is in the prevailing display mode. This command is useful for examining data structures that have been placed in a queue, or the operands of instructions.

To execute this command, press the Tab key.

This command changes the current address (.) to the location displayed.

This command does not affect the display mode.

Example

VAX

The following example applies only to OpenVMS VAX.

```
69B!BRB     0000067A     ❶ [Tab]
0000067A/CLRQ    -(SP)  ❷
```

❶ The instruction at 69B is displayed using the ! command. DELTA/XDELTA displays a branch instruction.

❷ The instruction at the address referred to by the branch instruction is displayed by pressing the Tab key. DELTA/XDELTA displays the instruction at address 67A.♦

Example

AXP

The following example applies only to OpenVMS AXP.

```
10000/00083089    ❶
00010004/00000000 ❷
00010008/00030000 ❸
00030000/23DEFFE0
```

❶ The contents of location 10000 are displayed using the / command.

❷ The subsequent two locations are displayed using the LINEFEED command.

❸ After displaying the contents of location 10008 (30000), the TAB command is used to display the contents of location 30000.♦

" (Open Location and Display Contents in ASCII)

Displays the contents of a location as an ASCII string.

Format

[pid:] start-addr-exp[,end-addr-exp] "

Arguments

pid
The internal process identification (PID) of a process you want to access. If you specify zero, or do not specify any PID, the default process is the current process. This argument cannot be used with XDELTA.

Subsequent open location and display contents commands issued after using the **pid** argument display the contents of the location of the specified process until you specify another PID with this command.

You can obtain the internal PID of processes by running the System Dump Analyzer utility (SDA). Use the SDA command SHOW SUMMARY to determine the external PID. Then use the SDA command SHOW PROCESS/INDEX to determine the internal PID. Refer to your operating system's *System Dump Analyzer Utility Manual* for more information about SDA commands.

start-addr-exp
The address of the location, or the start of a range of locations, to be displayed. If you want to view one location, the syntax is as follows:

start-add-exp"

end-addr-exp
The last address within a range of locations to be viewed. If you want to view a series of locations, the syntax is as follows:

start-add-exp,end-addr-exp"

Description

The Open Location and Display Contents in ASCII command opens the location or range of locations at **start-addr-exp** and displays the contents in ASCII format. This command does not change the width of the display (byte, word, longword) from the prevailing mode. If the prevailing mode is word mode, two ASCII characters are displayed; if byte mode, one character is displayed.

The display mode remains ASCII until you enter the next Open Location and Display Contents command (/) or Open Location and Display Contents in Instruction Mode command (!). These commands change the display mode to hexadecimal or instruction, respectively.

You can modify the contents of the locations, starting at **start-addr-exp**, with the Deposit ASCII string (') command.

" (Open Location and Display Contents in ASCII)

Example

```
235FC2 [W/415A    ❶
235FC2" ZA        ❷  Linefeed  ❸
235FC4/PP
```

❶ The current display mode is word (displays one word in hexadecimal).

❷ The " command changes the prevailing display mode to ASCII but does not affect the width of the display.

❸ The next Close Current Location, Open Next command (LINEFEED) determines the address of the location to open by adding the width, in bytes, to the value contained in the symbol . (the current address). Then it opens the number of bytes equal to the width of the prevailing display mode, which in this example is two bytes.

The ASCII representation of the contents of the location presents the bytes left to right, while the hexadecimal representation presents them right to left.

RETURN (Close Current Location)

Closes a location that has been opened by one of the open location and display contents commands.

Format

RETURN

Description

If you have opened a location with one of the open location and display contents commands (/, LINEFEED, ESC, TAB, !, or "), press the Return key to close the location. Use this command to make sure that a specific location has not been left open with the possibility of being overwritten.

You also press the Return key to terminate the following DELTA/XDELTA commands:

- ;X
- ;E
- ;G
- ;P
- ;B
- ;M
- 'string'
- ;L
- EXIT (DELTA only)

AXP

On AXP, the same is true for the commands that are specific to this implementation, as follow:

- ;Q
- ;C
- ;W
- ;I
- ;H
- \string\ ♦

On AXP and VAX, you can also use the Return key as an ASCII character in a quoted string. Refer to the Deposit ASCII String command (').

;B (Breakpoint)

Shows, sets, and clears breakpoints.

Format

[addr-exp][,n][,display-addr-exp][,cmd-string-addr] **;B**

Arguments

addr-exp
The address where you want the breakpoint.

n
The number to assign to the breakpoint. If you omit a number, DELTA/XDELTA assigns the first unused number to the breakpoint; if all numbers are in use, DELTA/XDELTA displays the error message, "EH?".

VAX

On VAX, for XDELTA, the range is from 2 to 8. For XDELTA, breakpoint 1 is reserved for INI$BRK. For DELTA, the range is from 1 to 8.♦

AXP

On AXP, for XDELTA, the range is from 1 to 8. For DELTA, the range is from 1 to 8.♦

display-addr-exp
The address of a location, the contents of which are to be displayed in hexadecimal in the prevailing width mode when the breakpoint is encountered. Omit this argument by specifying zero or two consecutive commas. If omitted, DELTA/XDELTA displays only the instruction that begins at the specified address.

cmd-string-addr
The address of the string of DELTA/XDELTA commands to execute when this breakpoint is encountered. Refer to the Execute Command String (;E) command. DELTA/XDELTA displays the information requested before executing the string of commands associated with complex breakpoints. You must have previously deposited the string of commands using the ' command or have coded the string into an identifiable location in your program. If omitted, DELTA/XDELTA executes no commands automatically and waits for you to enter commands interactively.

Description

The breakpoint command shows, sets, and clears breakpoints. The action of this command depends on the arguments used with it. Each action is described below.

Displaying Breakpoints

To show all the breakpoints currently set, enter ;B. For each breakpoint, DELTA /XDELTA displays the following information:

- Number of the breakpoint
- Address of the breakpoint
- Address of a location the contents of which will be displayed when the breakpoint is encountered

- Address of the command string associated with this breakpoint (for complex breakpoints, refer to the section in this Description called Setting Complex Breakpoints)

Setting Simple Breakpoints

To set a breakpoint, enter an address expression followed by ;B. Then press the Return key, as follows:

addr-exp;B Return

DELTA/XDELTA sets a breakpoint at the specified location and assigns it the first available breakpoint number.

When DELTA/XDELTA reaches the breakpoint, it completes the following actions:

- Suspends instruction execution.
- Sets a flag to change the display mode to instruction mode. Any subsequent Close Current Location, Open Next (LINEFEED) commands and Open and Display Indirect Location (TAB) commands will display locations as machine instructions.
- VAX — On VAX, the following message is displayed, listing the number of the breakpoint, the address of the breakpoint, and the instruction stored at the breakpoint location:

```
n BRK at address
address/decoded-instruction♦
```

AXP — On AXP, the format of the display differs slightly, as shown in the following example:

```
 Brk n at address
address!decoded-instruction♦
```

If you are using XDELTA in a multiprocessor environment, the CPU ID of the processor where the break was taken is also displayed.

VAX — On VAX, the CPU ID is displayed as a 2-digit hexadecimal number. ♦

AXP — On AXP, the CPU ID is displayed as a decimal number with no leading zeros.♦

On AXP and VAX, after the breakpoint message is displayed, you can enter other DELTA/XDELTA commands. You can reset the flag that controls the mode in which instructions are displayed by entering the Open Location and Display Contents (/) command.

Setting a Breakpoint and Assigning a Number to It

To set a breakpoint and assign it a number, enter the address where you want the breakpoint, a comma, a single digit for the breakpoint number, a semicolon (;), the letter B, and then press the Return key.

For example, if you wanted to set breakpoint 4 at address 408, the command is as follows:

```
408,4;B Return
```

DELTA/XDELTA sets a breakpoint at the specified location and assigns it the specified breakpoint number.

Clearing Breakpoints

To clear a breakpoint, enter zero (0), followed by a comma, the number of the breakpoint to remove, a semicolon (;), the letter B, and then press the Return key. DELTA/XDELTA clears the specified breakpoint. For example, if you wanted to clear breakpoint 4, the command is as follows:

```
0,4;B Return
```

VAX

On VAX, when using XDELTA, do not clear breakpoint 1. If you do, any calls to INI$BRK in your program will not result in entry into XDELTA.♦

Setting Complex Breakpoints

On AXP and VAX, a complex breakpoint completes one or more of the following actions:

- Always displays the next instruction to be executed
- Optionally displays the contents of another, specified location
- Optionally executes a string of DELTA/XDELTA commands stored in memory

To use the complex breakpoint, you must first create the string of DELTA commands you want executed. Then deposit those commands at a memory location with the Deposit ASCII String command (').

To set a complex breakpoint, use the following syntax:

addr-exp,n,display-addr-exp,cmd-string-addr;B

Example

```
;B
1  00000690
2  00000699                    ❶
0,2;B                          ❷
;B
1  00000690                    ❸
;P ❹
```

❶ Two breakpoints have already been set and are displayed. Using ;B, DELTA /XDELTA displays each breakpoint number and the address location of each breakpoint.

❷ Breakpoint 2 is cleared.

❸ Current breakpoints are displayed. Because breakpoint 2 has been cleared, DELTA/XDELTA displays just breakpoint 1.

❹ Program execution is continued using the ;P command.

;P (Proceed from Breakpoint)

Continue program execution following a breakpoint.

Format

;P

Description

The Proceed from Breakpoint command continues program execution at the address contained in the PC of the program. Program execution continues until the next breakpoint or until program completion.

AXP

If DELTA/XDELTA does not have write access to the target of a JSR instruction, you cannot use the S or ;P command at the JSR instruction. First, you must use the O command; then you can use the S or ;P command.♦

Example

VAX

The following example applies only to OpenVMS VAX.

```
;B
2  00000699                    ❶
;P                             ❷
2 BRK AT 00000699
00000699/BSBB    000006A2      ❸
```

❶ Current breakpoints are displayed using ;B (breakpoint 2 at address 699).

❷ Program execution is continued using the ;P command.

❸ Program execution halts at breakpoint 2. DELTA/XDELTA displays the breakpoint message (the breakpoint number and the address) and the instruction.♦

Example

AXP

The following example applies only to OpenVMS AXP.

```
;B
 1 00030010        ❶
;P        ❷
Brk 1 at 00030010
00030010!       STQ             R26,#X0008(SP)  ❸
```

❶ Current breakpoints are displayed using ;B (breakpoint 1 at address 30010).

❷ Program execution is continued using the ;P command.

❸ Program execution halts at breakpoint 1. DELTA/XDELTA displays the breakpoint message (the breakpoint number and the address) and the instruction.♦

;G (Go)

Continues program execution.

Format

address-expression **;G**

Parameters

address-expression
The address at which to continue program execution.

Description

The Go command places the address you specified in **address-expression** into the PC and continues execution of the program at that address. It is useful when you want to ignore specific lines of code or return to a previous program location to repeat execution.

Example

```
6A2;G
```

Program execution is started at address 6A2.

S (Step Instruction)

Executes one instruction and displays the next. If the executed instruction is a call to a subroutine, it steps into the subroutine and displays the next instruction to be executed in the subroutine.

Format

S

Description

The Step Instruction command executes one instruction and displays the next instruction (in instruction mode) and its address. Use this command to single-step instructions, including single-stepping all instructions in subroutines. If you want to exclude single-stepping instructions in subroutines, use the O command.

The instruction displayed has not yet been executed. This command sets a flag to change the display mode to instruction mode. Any subsequent Close Current Location, Open Next (LINEFEED) commands and Open and Display Indirect Location (TAB) commands will display locations as machine instructions. The Open Location and Display Contents (/) command clears the flag, causing the display mode to revert to longword, hexadecimal mode.

VAX

On VAX, if the instruction being executed is a BSBB, BSBW, JSB, CALLG, or CALLS instruction, Step moves to the subroutine called by these instructions and displays the first instruction within the subroutine.♦

AXP

On AXP, if the instruction being executed is a JSR or BSR instruction, Step moves to the subroutine called by these instructions and displays the first instruction within the subroutine.

Note

If DELTA/XDELTA does not have write access to the target of a JSR instruction, you cannot use the S or ;P command at the JSR instruction. First, you must use the O command; then you can use the S or ;P command.♦

On AXP and VAX, in general, you move to the instruction where you want to start single-step execution by placing a breakpoint at that instruction and typing ;P. Then press S to execute the first instruction and display the next one.

Example

VAX

The following example applies only to OpenVMS VAX.

```
00000690/CMPL    R0,#000009A8    S    ❶
00000697/BEQL    0000069D        S    ❷
00000699/BSBB    000006A2        S    ❸
000006A2/PUSHL   R2                   ❹
```

❶ Step program execution is started at address 690. The instruction at 690 is executed and the next instruction is displayed. Step execution is continued using S.

❷ At address 697, there is a branch instruction to the instruction at address 69D. However, because the condition (BEQL) is not met, program execution continues at the next instruction. The next S command is executed.

❸ At address 699, there is a branch instruction to the instruction at address 6A2, a subroutine. The next S command is executed.

❹ Program execution moves to the subroutine.♦

Example

AXP

The following example applies only to OpenVMS AXP.

```
0003003C!       BLBC            R0,#X000006 S   ❶
00030040!       LDQ             R16,#X0050(R2) S ❷
00030044!       BIS             R31,R31,R17 S   ❸
00030048!       LDQ             R26,#X0040(R2)
```

❶ Step program execution is started at address 3003C. The instruction at 3003C is a conditional branch instruction. Step execution is continued using the S command.

❷ Because the condition (BLBC) was not met, program execution continued at the next instruction at address 30040. Had the branch been taken, execution would have continued at address 30058. The second S command causes the LDQ instruction to be executed.

❸ The instruction at address 30044 is displayed. The S command is executed.♦

O (Step Instruction over Subroutine)

Executes one instruction, steps over a subroutine by executing it, and displays the instruction to which the subroutine returns control.

Format

O

Description

The Step Instruction over Subroutine command executes one instruction and displays the address of the next instruction. If the instruction executed is a call to a subroutine, the subroutine is executed and the next instruction displayed is the instruction to which the subroutine returns control. Use this command to do single-step instruction execution excluding single-stepping of instructions within subroutines. If you want to do single-step execution of all instructions, including those in subroutines, use the S command.

This command sets a flag to change the display mode to instruction mode. Any subsequent Close Current Location, Open Next (LINEFEED) commands and Open and Display Indirect Location (TAB) commands will display locations as machine instructions. The Open Location and Display Contents (/) command clears the flag, causing the display mode to revert to longword, hexadecimal mode.

VAX

On VAX, the subroutine call instructions are BSBB, BSBW, JSB, CALLG, and CALLS.♦

AXP

On AXP, the subroutine call instructions are JSR and BSR.♦

On AXP and VAX, if you set a breakpoint in the subroutine and enter the O command, program execution breaks at the subroutine breakpoint. When you enter a Proceed command (;P), and program execution returns to the instruction to which the subroutine returns control, a message is displayed, as follows:

```
STEPOVER BRK AT nnnnnnnn
instruction
```

The message informs you that program execution has returned from a subroutine.

If you are using XDELTA in a multiprocessor environment, the CPU ID of the processor where the break was taken is also displayed.

VAX

On VAX, the CPU ID is displayed as a 2-digit hexadecimal number.♦

AXP

On AXP, the CPU ID is displayed as a decimal number with no leading zeros.♦

Example

VAX

The following example applies only to OpenVMS VAX.

```
6D5;B                                                    ❶
;P                                                       ❷
1 BRK AT 000006D5
000006D5/CALLS    #0C,@#7FFEDE00                ;P     ❸
        PID=  0006      LOGINTIME=  12:50:29.45
2 BRK AT 00000699
00000699/BSBB    000006A2       ;P                      ❹
1 BRK AT 000006D5
000006D5/CALLS    #0C,@#7FFEDE00                ;P     ❺
        PID=  0007      LOGINTIME=  12:50:37.08
2 BRK AT 00000699
00000699/BSBB    000006A2       O                       ❻
1 BRK AT 000006D5
000006D5/CALLS    #0C,@#7FFEDE00                ;P     ❼
        PID=  0008      LOGINTIME=  12:50:45.64
STEPOVER BRK AT 0000069B                                 ❽
0000069B/BRB    X1+047A
```

❶ One breakpoint has been set at address 699 in the main routine. A simple breakpoint is set at 6D5 using ;B. This breakpoint is in a subroutine.

❷ Program execution continues using ;P.

❸ Program execution stops at breakpoint 1, which is in the subroutine. DELTA /XDELTA displays the breakpoint message and the instruction at the new breakpoint. Program execution continues using ;P.

❹ The subroutine completes and displays some output. Program execution continues until breakpoint 2. DELTA/XDELTA displays the breakpoint message and the breakpoint 2 instruction. Program execution continues with the ;P command.

❺ Program execution stops at breakpoint 1. Program execution continues with the ;P command. The subroutine completes execution and displays the output.

❻ Program execution stops at breakpoint 2. The subroutine is stepped over to the next instruction using the O command.

❼ Program execution stops at breakpoint 1 in the subroutine. Program execution continues using the ;P command.

❽ The subroutine completes execution and displays output. DELTA/XDELTA displays a STEPOVER break message that states the O command has been completed, returning control at address 69B.♦

Example

AXP

The following example applies only to OpenVMS AXP.

```
30040;B          ❶
30070;B          ❷

;B
 1 00030040
 2 00030070
```

```
;P                    ❸
Brk 1 at 00030040

00030040!       LDA             R27,#XFFC8(R2) O ❹
00030044!       BSR             R26,#X00000A O   ❺
Brk 2 at 00030070

00030070!       LDA             SP,#XFFD0(SP) ;P ❻

Step-over at 30048
00030048!       LDQ             R26,#X0048(R2) S ❼
0003004C!       BIS             R31,R31,R17
```

❶ A simple breakpoint is set in the main routine at address 30040, just prior to the subroutine call.

❷ A simple breakpoint is set in the subroutine at address 30070. The breakpoints are displayed using the ;B command.

❸ Program execution continues using ;P.

❹ Program execution stops at breakpoint 1. DELTA/XDELTA displays the breakpoint message and the instruction at the breakpoint address. The O command is used to single-step (DELTA/XDELTA recognizes that this is not a call instruction and turns it into a single-step instead).

❺ The next instruction is a subroutine call (BSR). The subroutine is stepped over using the O command.

❻ Ordinarily, the step-over would continue execution at the instruction following the subroutine call. However, in this case, program execution stops at breakpoint 2 inside the subroutine at address 30070. Program execution continues with the ;P command.

❼ The subroutine completes execution. DELTA/XDELTA displays a step-over break message that indicates that the O command has been completed, returning control at address 30048.♦

' (Deposit ASCII String)

Deposits the ASCII string at the current address.

Format

'string'

Arguments

string
The string of characters to be deposited.

Description

The Deposit ASCII String command deposits **string** at the current location (.) in ASCII format. The second apostrophe is required to terminate the string. All characters typed between the first and second apostrophes are entered as ASCII character text. Avoid embedding an apostrophe (') within the string you want to deposit.

When you want to use key commands (LINEFEED, RETURN, ESC, or TAB), press the key. These commands are entered as text.

This command stores the characters in 8-bit bytes and increments the current address (.) by one for each character stored.

This command does not change the prevailing display mode.

Example

```
7FFE1600/'R0/ Linefeed Linefeed '
```

The ASCII string "R0/ Linefeed Linefeed " is stored at address 7FFE1600. This string, if subsequently executed with the ;E command, examines the contents of general register 0 (the command R0/), then examines two subsequent registers (using two LINEFEED commands).

;E (Execute Command String)

Executes a string of DELTA/XDELTA commands stored in memory.

Format

address-expression **;E**

Arguments

address-expression
The address of the string of DELTA/XDELTA commands to execute.

Description

The Execute Command String command executes a string of DELTA/XDELTA commands. Load the ASCII text command string to a specific location in memory using the Deposit ASCII String command (') or code the string in your program into an identifiable location.

If you want DELTA/XDELTA to proceed with program execution after it executes the string of commands, end the command string with the ;P command. If you want DELTA/XDELTA to wait for you to enter a command after it executes the string of commands, end the command string with a null byte (a byte containing 0).

XDELTA, but not DELTA, provides two command strings in memory.

VAX On VAX, the addresses of these command strings are stored in base registers XE and XF. The string addressed by XE displays the physical page number (PFN) database for the PFN in X0. The string addressed by XF copies the PFN in R0 to base register X0. It then displays the PFN database for that PFN.♦

AXP On AXP, the addresses of these command strings are stored in base registers X14 and X15. The string addressed by X14 displays the physical page number (PFN) database for the PFN in X0. The string addressed by X15 copies the PFN in R0 to base register X0. It then displays the PFN database for that PFN.♦

On AXP and VAX, you can use the command strings provided with XDELTA to obtain the following information:

- Specified PFN
- PFN state and type
- PFN reference count
- PFN backward link or working-set-list index
- PFN forward link or share count
- Page table entry (PTE) address that points to the PFN
- PFN backing-store address
- VAX On VAX, the virtual block number in the process swap image, the block to which the page's entry in the SWPVBN array points♦
- AXP On AXP, the virtual page number in process swap image, the collection of blocks containing the page as pointed to by the PFN database♦

Example

```
7FFE1600,0;X      ❶
7FFE1600          ❷
X0;E              ❸
R0/00000001       ❹
R1/00000000
R2/00000000
```

❶ The address (7FFE1600) where an ASCII string is stored is placed into base register 0 using ;X.

❷ DELTA/XDELTA displays the value in X0.

❸ The command string stored at address 7FFE1600, which is to examine the contents of R0, R1, and R2 (R0/Linefeed Linefeed), is executed with ;E.

❹ DELTA/XDELTA executes the commands and displays the contents of R0, R1, and R2.

;X (Load Base Register)

Places an address in a base register.

Format

address-expression,n[,y];**X**

Arguments

address-expression
The address to place in the base register.

n
The number of the base register.

AXP

y
On AXP, a parameter for modifying the default offset of 100000_{16}. The valid range is 1 to FFFFFFFF.♦

Description

VAX

On VAX, to place an address in a base register, enter an expression followed by a comma (,), a number from 0 to F_{16}, a semicolon (;), and the letter X.♦

AXP

On AXP, to place an address in a base register, enter an expression followed by a comma (,), or a number from 0 to 15_{10}, optionally, a number from 1 to FFFFFFFF, a semicolon (;), and the letter X.♦

On AXP and VAX, DELTA/XDELTA places the address in the base register. DELTA/XDELTA confirms that the base register is set by displaying the value deposited in the base register.

For example, the following command places the address 402 in base register 0. DELTA/XDELTA then displays the value in the base register to verify it.

```
402,0;X Return
00000402
```

Whenever DELTA/XDELTA displays an address, it will display a relative address if the address falls within the computer's valid range for an offset from a base register. The relative address consists of the base register identifier (Xn), followed by an offset. The offset gives the address location in relation to the address stored in the base register.

For example, if base register 2 contains 800D046A, the address that would be displayed is X2+C4, the base register identifier followed by the offset.

Relative addresses are computed for both opened and displayed locations and for addresses that are instruction operands.

If you have defined several base registers, the offset will be relative to the closest base register. If an address falls outside the valid range, it is displayed as a hexadecimal value.

VAX

On VAX, the default offset is 2000_{16} bytes. It cannot be modified.♦

AXP

On AXP, the default offset is 100000_{16}, which can be modified.♦

Example

VAX

The following example applies only to OpenVMS VAX.

```
00000664/CLRQ    -(SP)    200,1;X    ❶
00000200                             ❷

X1 490!CMPL    R0,#000009A8          ❸
X1 499!BSBB    X1+04A2               ❹
```

❶ The base address of the program (determined from the map file) is virtual address 200. The base address is stored in base register 1 with ;X.

❷ DELTA/XDELTA displays the value in base register 1 just loaded, 200.

❸ The instruction at offset 490 is displayed in instruction mode using the ! command. The address reference is X1+490 (the + sign is implied when not specified). DELTA/XDELTA displays the instruction at address X1+490.

❹ The instruction at offset 499 is displayed. This instruction is a branch instruction. DELTA/XDELTA displays the address of the branch in offset notation.♦

Example

AXP

The following example applies only to OpenVMS AXP.

```
30000,0;X       ❶
00030000
30070,1,200;X       ❷
00030070
;X          ❸
  0 00030000
  1 00030070  00000200

S          ❹
X0+00000004!    BIS             R31,R31,R18

x1+10!  STQ             FP,#X0020(SP)    ❺
```

❶ The base address of the program (determined from the map file) is virtual address 30000. The base address is stored in base register 0 with ;X, using the default offset. DELTA/XDELTA displays the value in base register 0 just loaded, 30000.

❷ The address of a subroutine, 30070, is stored in base register 1, specifying a new offset of 200 (to override the default value of 100000). Note that this command could also have been expressed as "x0+70,1,200;X". DELTA /XDELTA displays the value in base register 1 just loaded, 30070.

❸ The ;x command is used to display the current base registers. Note that for those not using the default offset, the offset is also displayed.

❹ The S command is used to execute the first instruction in the main routine. DELTA/XDELTA displays the address of the next instruction, 30004, as x0+00000004 and then displays the instruction at that address.

❺ The instruction at offset 10 from base register 1 is displayed in instruction mode using the ! command.♦

= (Display Value of Expression)

Evaluates an expression and displays its value.

Format

expression =

Argument

expression
The expression to be evaluated.

Description

The Display Value of Expression command evaluates an expression and displays its value in hexadecimal. The expression can be any valid DELTA/XDELTA expression. See Section 2.1 for a description of DELTA/XDELTA expressions.

All calculations and displays are in hexadecimal in the prevailing length mode.

Example

```
FF+1=00000100  ❶
A-1=00000009   ❷
```

❶ FF_{16} and 1_{16} are added together. DELTA/XDELTA displays the sum in hexadecimal.

❷ 1_{16} is subtracted from A_{16}. DELTA/XDELTA displays the result in hexadecimal.

;M (Set All Processes Writable)

Sets the address spaces of all processes to be writable or read-only by your DELTA process. This command can be used only with DELTA. Use of this command requires CMKRNL privilege.

AXP

On AXP, this command also sets writable the general purpose registers of other processes, if, after issuing the ;M command, you specify another process with any command that takes the PID argument, such as the / command.♦

Format

n;**M**

Argument

n
Specifies your process privileges for reading and writing at other processes. If 0, your DELTA process can only read locations in other processes; if 1, your process can read or write any location in any process. If not specified, DELTA returns the current value of the M (modify) flag (0 or 1).

Description

The Set All Processes Writable command is useful for changing values in the running system.

Note

Use this activity very carefully during timesharing. It affects all processes on the system. For this reason, your process must have change-mode-to-kernel (CMKRNL) privilege to use this command. It is safest to use this command only on a standalone system.

;L (List Names and Locations of Loaded Executive Images)

List the names and virtual addresses of all loaded executive images.

Format

[sequence number]**;L**

Argument

sequence number

AXP

On AXP, specifies a single executive image.♦

Description

On AXP and VAX, use the ;L command when you are debugging code that resides in system space. Although you use this command mostly with XDELTA, you can use it with DELTA if your process has change-mode-to-executive (CMEXEC) privilege and you are running a program in executive mode.

This command lists the names and locations of the loaded modules of the executive. A loading mechanism maps a number of images of the executive into system space. The ;L command lists the currently loaded images with their starting and ending virtual addresses. If you enter ;L before all the executive images are loaded (for example, at an XDELTA initial breakpoint), only those images that have been loaded will be displayed.

AXP

On AXP, this command displays additional information and provides a second use, based on the additional information. For each loaded executive image that is sliced into discontiguous image sections, the display shows the sequence number for the executive image and the base and ending addresses of each image section. A second use of this command is to display the base and ending addresses of a single image if you specify its sequence number.♦

Examples

VAX

The following example applies only to OpenVMS VAX. It shows the names and the starting and ending virtual addresses of the three executive images that are loaded in memory.

```
;L
PRIMITIVE_IO.EXE                800EAA00  800EBC00
SYSTEM_SYNCHRONIZATION.EXE      800EBC00  800ED400
SYSTEM_PRIMITIVES.EXE           800ED400  800F1000♦
```

AXP

The following examples apply only to OpenVMS AXP.

In the following example, the names, the starting and ending virtual addresses, and the sequence numbers for all the loaded executive images are shown. Only one image, EXEC_INIT.EXE, was not split into image sections. For every image that was split into image sections, it also shows the name and the base and ending address of each section.

;L (List Names and Locations of Loaded Executive Images)

```
;L

Seq#  Image Name                                Base      End

0012  EXEC_INIT.EXE                             8080C000  80828000
0010  SYS$CPU_ROUTINES_0101.EXE
          Nonpaged read only                    80038000  8003A200
          Nonpaged read/write                   80420200  80420A00
          Initialization                        80808000  80808400
000E  ERRORLOG.EXE
          Nonpaged read only                    8002E000  80036600
          Nonpaged read/write                   8041BE00  80420200
          Initialization                        80804000  80804800
000C  SYSTEM_SYNCHRONIZATION.EXE
          Nonpaged read only                    80024000  8002C800
          Nonpaged read/write                   8041A000  8041BE00
          Initialization                        80800000  80800800
                .       .     .
                .       .     .
                .       .     .
0002  SYS$BASE_IMAGE
          Nonpaged read only                    80002000  80009400
          Nonpaged read/write                   80403000  80414C00
          Fixup                                 80620000  80620600
          Symbol Vector                         8040B010  80414560
0000  SYS$PUBLIC_VECTORS.EXE
          Nonpaged read only                    80000000  80001C00
          Nonpaged read/write                   80400000  80403000
          Fixup                                 8061E000  8061E200
          Symbol Vector                         80401BF0  80402ED0
```

The following example illustrates the use of the sequence number with the ;L command to display information about one image. In this example, the sequence number C for the SYSTEM_SYNCHRONIZATION.EXE module is specified with the ;L command. (It is not necessary to specify the leading zeros in the command.) The resulting display shows only the SYSTEM_SYNCHRONIZATION.EXE module (whose sequence number is 000C). The display includes the names of the image sections within the module and their base and ending addresses.

```
C;L

Seq#  Image Name                                Base      End

000C  SYSTEM_SYNCHRONIZATION.EXE
          Nonpaged read only                    80024000  8002C800
          Nonpaged read/write                   8041A000  8041BE00
          Initialization                        80800000  80800800♦
```

;Q (Validate Queue) (AXP Only)

AXP

Analyzes absolute and self-relative longword queues and displays the results of the analysis.

Format

queue_header_address[,queue_type];**Q**

Argument

queue_header_address
The queue header must be at least longword aligned.

queue_type
A queue type of zero (the default) represents an absolute queue. A queue type of 1 indicates a self-relative queue.

Description

The validate queue function is similar to the one in the OpenVMS AXP System Dump Analyzer Utility. It can analyze both absolute and self-relative longword queues and display the results of the analysis. This function identifies various problems in the queue headers and invalid backward links for queue entries and evaluates the readability of both. For valid queues, it tells you the total number of entries. For invalid queues, it tells you the queue entry number and the address that is invalid and why.

Example

```
FFFFFFFF8000F00D;Q      !Absolute at GF00D
GF00D,0;Q               !Absolute at GF00D
GF00,1;Q                !Self-relative at GF00♦
```

;C (Force the system to bugcheck and crash) (AXP Only)

AXP

Force the system to bugcheck and crash.

Format

;C

Description

The ;C command forces the system to bugcheck and crash. You can do this from wherever you are in your debugging session. Although this command is for use primarily with XDELTA, you can also use it with DELTA, but only in kernel mode. When you issue this command, the following message is generated:

```
BUG$_DEBUGCRASH, Debugger forced system crash♦
```

;W (List Name and Location of a Single Loaded Image) (AXP Only)

AXP

Lists information about an image that contains the address you supplied.

Format

address-expression **;W**

sequence number, offset**;W**

Arguments

address-expression
An address contained within an executive image or a user image.

sequence number
The identifier assigned to an executive image.

offset
The distance from the base address of the image.

Description

The ;W command is used for debugging code that resides in system or user space. You can use this command with XDELTA for debugging an executive image. You can also use this command with DELTA.

To examine the executive image list, you must be running in executive mode or your process must have change-mode-to-executive (CMEXEC) privilege.

This command can be used in two ways. In the first way, if you supply an address that you are trying to locate, the command lists the name of the executive or user image that contains the address, its base and ending addresses, and the offset of the address from the base of the image. For any executive image that has been sliced, it also displays its sequence number. The offset can be used with the link map of the image to locate the actual code or data.

In the second way, if you supply the sequence number of a sliced executive image and an offset, the command computes and displays the current location in memory.

Examples

The first form of the command takes a system space address as a parameter and attempts to locate that address within the loaded executive images. This command works for both sliced and unsliced loadable executive images. The output is very similar to ;L, except the offset is displayed for you, as shown in the following example:

```
80026530;W
Seq#  Image Name                          Base      End       Image Offset
000C  SYSTEM_SYNCHRONIZATION.EXE
          Nonpaged read only               80024000  8002C800  00002530
```

;W (List Name and Location of a Single Loaded Image) (AXP Only)

The second form of the command takes a loadable executive image sequence number and an image offset from the map file as parameters. The output, again, is very similiar to ;L, except that the system space address that corresponds to the image offset is displayed, as shown in the following example:

```
C,2530;W
Seq#  Image Name                              Base      End       Address
000C  SYSTEM_SYNCHRONIZATION.EXE
          Nonpaged read only                  80024000  8002C800  80026530♦
```

;I (List Information About the Current Main Image and Its Shareable Images)(AXP Only)

AXP

List information about the current main image and all shareable images that were activated, including those that were installed /RESIDENT.

Format

;I

Description

The ;I command peruses the image control block (IMCB) list and displays information about the current main image and all shareable images that were activated, including those that were installed /RESIDENT. The ;I command differs from the ;L command which displays information about the loadable image database.

The display of the ;I command is similar to the ;L command display. It shows the image name, the starting and ending addresses, the symbol vector address, and some flags. The command is useful for debugging shareable images. For example, the display enables you to determine where LIBRTL is mapped.

The field flags are M, S, and P. The flag M indicates the main image; S or P indicates images that are installed as shareable or protected, respectively.

Unlike the ;L command, which only works from kernel mode or when you have CMEXEC or CMKRNL privileges, the ;I command works from any mode. However, to modify the IMCB database, you must be in executive or kernel mode.

For sliced main and shareable images, the ;I command also includes an entry for each resident code section and each compressed data section, which shows the base and end address for each section.

The ;I command is implemented only for DELTA.

Example

```
$ define lib$debug delta
$ run/debug hello
Alpha/VMS DELTA Version 1.5

Brk 0 at 00020040

00020040!     LDA                SP,#XFFD0(SP) ;i
Image Name                          Base     End      Symbol-Vector  Flags

HELLO                               00010000 000301FF                M
DECC$SHR                            00032000 001233FF 00106B90       S
DPML$SHR                            0012C000 001AC5FF 0019DED0       S
LIBRTL                              001AE000 0025E7FF 00240790       S
Resident Code Sections:
                                    8015A000 801BBA00
LIBOTS                              00124000 0012A1FF 00128000       S
```

;I (List Information About the Current Main Image...)

```
Resident Code Sections:
                                    801BC000 801C2C00
Compressed Data Sections:
                                    00124000 00124A00
                                    00126000 00126800
                                    00128000 00128600
                                    0012A000 0012A200
SYS$PUBLIC_VECTORS                  80401C98 80403028 80401C98
DELTA                               00260000 002943FF 00260000
SYS$BASE_IMAGE                      8040C5B0 804163E0 8040C5B0♦
```

;H (Video Terminal Display Command) (AXP Only)

AXP

Specifies the display mode, either hardcopy terminal mode or DEC-CRT.

Format

;H

Description

The ;H command enables you to choose the display mode of DELTA/XDELTA output. You can display output either in hardcopy terminal mode or DEC-CRT mode. The default display is DEC-CRT mode. You can toggle back and forth from one display mode to the other by repeating the ;H command.♦

string\ (Immediate mode text display command) (AXP Only)

AXP

Displays the ASCII text string enclosed in backslashes.

Format

string\

Description

This mode is useful when creating your own predefined command strings. Use the backslash to begin and end an ASCII text string. Follow the ending backslash with a terminator. When DELTA or XDELTA encounters the ending backslash and terminator, it prints the ASCII text string.♦

EXIT (Exit from DELTA Debugging Session)

Terminates the DELTA debugging session. Use with DELTA only.

Format

EXIT

Description

Use the EXIT command to terminate a DELTA debugging session. You cannot use EXIT in XDELTA.

You may have to enter EXIT twice, such as when your program terminates execution via the $EXIT system service or via the Return key (to DCL).

A

Sample DELTA Debug Session on VAX

This appendix gives an example of using DELTA to debug a program on OpenVMS VAX. The program, LOGINTIM, uses the system service SYS$GETJPI to obtain the login times of each process. Although this is an example of using DELTA, most of the commands in the example could be used in an XDELTA debugging session.

To run this program without error, you need WORLD privilege.

The .LIS file is listed in Example A–1. Only the offsets and source code are shown.

Example A–1 Program for Getting LOGINTIMs

```
0000     1 ;++
0000     2 ; This sample program uses the wildcard feature of GETJPI to get the
0000     3 ; LOGINTIM for each active process.  It outputs the PID and LOGINTIM
0000     4 ; for each and exits when there are NOMOREPROCs.
0000     5 ;--
0000     6
0000     7 ;
0000     8 ; Data areas.
0000     9 ;
0000    10 DEVNAM: .ASCID /SYS$OUTPUT/          ;Output device specifier
000E
0012    11
0012    12 CHAN:   .LONG  0                     ;Assigned output channel
0016    13
0016    14 ITMLST:                              ;Item list for GETJPI call
0016    15         .WORD  8                     ; Byte length of output buffer
0018    16         .WORD  JPI$_LOGINTIM         ; Specify LOGINTIM item code
001A    17         .ADDRESS      TIME           ; Address of output buffer
001E    18         .LONG  0                     ; Not interested in return length
0022    19         .LONG  0                     ;Item list terminator
0026    20
0026    21 TIME:   .QUAD  0                     ;Buffer to hold LOGINTIM
002E    22
002E    23 OUTLEN: .LONG  0                     ;FAO buffer length
0032    24 OUTBUF: .LONG  1024                  ;FAO buffer descriptor
0036    25         .ADDRESS BUF
003A    26 BUF:    .BLKB  1024                  ;FAO buffer
043A    27
043A    28 CTRSTR: .ASCID *!/!_PID= !XW!_LOGINTIME= !%T* ;FAO control string
0448
0454
045E    29
045E    30 PIDADR: .LONG  -1                    ;Wildcard PID control longword
0462    31
0462    32 ;++
0462    33 ; Start of program.
```

(continued on next page)

Example A–1 (Cont.) Program for Getting LOGINTIMs

```
0462     34 ;--
0462     35 S:       .WORD   0                   ;Entry mask
0464     36          $ASSIGN_S   DEVNAM,CHAN     ;Assign output channel
0475     37          MOVAB   TIME,R2             ;Load pointer to LOGINTIM
047A     38                                      ;     output buffer
047A     39 LOOP:    $GETJPI_S   ITMLST=ITMLST,-;Get LOGINTIM for a process
047A     40                  PIDADR=PIDADR
0490     41          CMPL    R0,#SS$_NOMOREPROC ;Are we done?
0497     42          BEQL    5$                  ;If EQL yes
0499     43          BSBB    GOT_IT              ;Process data for this process
049B     44          BRB     LOOP                ;Look for another process
049D     45
049D     46 5$:      MOVZBL  #SS$_NORMAL,R0      ;Set successful completion code
04A1     47          RET                         ;Return, no more processes
04A2     48
04A2     49 GOT_IT:  $FAO_S  CTRSTR,-            ;Format the output data
04A2     50                  OUTLEN,-
04A2     51                  OUTBUF,-
04A2     52                  PIDADR,R2
04B9     53          $QIOW_S CHAN=CHAN,-         ;Output to SYS$OUTPUT
04B9     54                  FUNC=#IO$_WRITEVBLK,-
04B9     55                       P1=BUF,-
04B9     56                       P2=OUTLEN
04DC     57           RSB                        ;Done with this process data
04DD     58
04DD     59            .END S
```

The .MAP file is listed in Example A–2. Only the Program Section Synopsis with the PSECT, MODULE, base address, end address, and length are listed.

Example A–2 LOGINTIM Program .Map File

```
                        +--------------------------+
                        ! Program Section Synopsis !
                        +--------------------------+

Psect Name      Module Name       Base     End          Length
----------      -----------       ----     ---          ------
. BLANK .                        00000200 000006E2 000004E3 (      1251.)
                .MAIN.           00000200 000006E2 000004E3 (      1251.)
```

The DELTA debugging session is listed in Example A–3.

Example A–3 DELTA Debugging Session Example

```
$ DEFINE LIB$debugging SYS$LIBRARY:DELTA                                    ❶
$ RUN/debugging LOGINTIM                                                    ❷
DELTA Version 6.0

00000664/CLRQ    -(SP)    200,1;X                                           ❸
00000200                                                                    ❹

X1 490!CMPL    R0,#000009A8    .;B                                          ❺
X1 499!BSBB    X1+04A2         .;B                                          ❻
;P                                                                          ❼

1 BRK AT 00000690
X1+0490/CMPL    R0,#000009A8  R0/00000001 ;P                                ❽
2 BRK AT 00000699
X1+499/BSBB    X1+04A2       O                                              ❾
        PID=  0000      LOGINTIME=  00:00:00.00                             ❿
X1+049B/BRB    X1+047A        ;P                                           ⓫
1 BRK AT 00000690
X1+0490/CMPL    R0,#000009A8  R0/00000001 ;P                                ⓬
2 BRK AT 00000699
X1+0499/BSBB    X1+04A2       O                                             ⓭
        PID=  0001      LOGINTIME=  00:00:00.00
X1+049B/BRB    X1+047A        ;P
1 BRK AT 00000690
X1+0490/CMPL    R0,#000009A8                                                ⓮
;B                                                                          ⓯
1  00000690
2  00000699                                                                 ⓰
0,1;B                                                                       ⓱
;B                                                                          ⓲
2  00000699                                                                 ⓳
;P                                                                          ⓴
2 BRK AT 00000699
X1+0499/BSBB    X1+04A2       O
        PID=  0004      LOGINTIME=  12:50:20.40
X1+049B/BRB    X1+047A        ;P                                           ㉑
2 BRK AT 00000699
X1+0499/BSBB    X1+04A2       ;P
        PID=  0005      LOGINTIME=  12:50:25.61                             ㉒
2 BRK AT 00000699
X1+0499/BSBB    X1+04A2       X1 4B9!CLRQ    -(SP)                          ㉓
[Linefeed]                                                                  ㉔
X1+04BB/CLRQ    -(SP)  [Linefeed]
X1+04BD/PUSHL    X1+002E  [Linefeed]
X1+04C1/PUSHAL    X1+003A  [Linefeed]
X1+04C5/CLRQ    -(SP)  [Linefeed]
X1+04C7/PUSHL    #00  [Linefeed]
X1+04C9/MOVZWL    #0030;-(SP)  [Linefeed]
X1+04CE/MOVZWL    X1+0012,-(SP)  [Linefeed]
X1+04D3/PUSHL     #00  [Linefeed]
X1+04D5/CALLS    #0C,@#7FFEDE00              .;B                            ㉕
;B                                                                          ㉖
1  000006D5
2  00000699
```

(continued on next page)

Example A–3 (Cont.) DELTA Debugging Session Example

```
;P                                                              ❼
1 BRK AT 000006D5
X1+04D5/CALLS    #0C,@#7FFEDE00                ;P              ❽
        PID=  0006      LOGINTIME=  12:50:29.45
2 BRK AT 00000699
X1+0499/BSBB    X1+04A2        ;P                              ❾
1 BRK AT 000006D5
X1+04D5/CALLS    #0C,@#7FFEDE00                ;P              ❿
        PID=  0007      LOGINTIME=  12:50:37.08
2 BRK AT 00000699
X1+0499/BSBB    X1+04A2        O                               ⓫
1 BRK AT 000006D5
X1+04D5/CALLS    #0C,@#7FFEDE00                ;P              ⓬
        PID=  0008      LOGINTIME=  12:50:45.64
STEPOVER BRK AT 0000069B                                        ⓭
X1+049B/BRB    X1+047A                         ;B              ⓮
1  000006D5
2  00000699                                                     ⓯
0,2;B                                                           ⓰
0,1;B                                                           ⓱
;B                                                              ⓲
;P                                                              ⓳
        PID=  0009      LOGINTIME=  12:51:22.51
        PID=  000A      LOGINTIME=  12:51:30.26
        PID=  000B      LOGINTIME=  12:51:36.21
        PID=  000C      LOGINTIME=  12:51:58.86                 ⓴
EXIT 00000001                                                   ㉑
80187E7E/POPR    #03                          EXIT              ㉒
```

❶ DELTA is enabled as the debugger.

❷ The example program LOGINTIM is invoked with DELTA.

❸ DELTA displays a version number and the first executable instruction. The base address of the program (determined from the map file) is virtual address 200. The base address is placed in base register 1 with ;X. Now references to an address can use the address offset notation. For example, a reference to the first instruction is X1+464 (or base address 200 + offset 464). Also, DELTA displays some address locations as offsets to the base address.

❹ DELTA displays the value in base register 1, just loaded 200.

❺ The instruction at address 690 is displayed in instruction mode using !. Its address location is expressed as the base address plus an offset. In the listing file, the offset is 490. The base address in base register X1 is 200. The address reference, then, is X1+490. (Note that the + sign is implied when not specified.)

A simple breakpoint is set at that address using the ;B command. The address reference for ;B is the . symbol, representing the current address. X1+490;B would have done the same thing.

❻ The same commands (! command to view the instruction and ;B to set a breakpoint) are repeated for the instruction at offset 499. When DELTA displays the instruction (BSBB GOT_IT), it displays the destination of the branch (GOT_IT) as the address location. DELTA displays the value as an offset to base register 1.

❼ Program execution is begun using ;P.

❽ Program execution halts at the first breakpoint. DELTA displays the breakpoint message (1 BRK AT 00000690) with the breakpoint number 1 and the virtual address. The virtual address is 00000690, which is the base address (200) plus the offset 490. DELTA then displays the instruction in instruction mode (CMPL R0,#000009A8). The contents of general register 0 are displayed with the / command. DELTA displays the contents of R0, which is 1. Program execution continues using the ;P command.

❾ Program execution halts at breakpoint 2. DELTA displays the breakpoint message, then the instruction. Step-instruction execution, excluding instructions in subroutines, is initiated with O.

❿ The subroutine GOT_IT is executed, and the output (PID and login time) is displayed.

⓫ The O command halts program execution at the instruction where the subroutine returns control (BRB LOOP). DELTA displays the instruction in instruction mode (BRB X1+047A), where X1+047A is the address of the first instruction in LOOP. Program execution continues with ;P.

⓬ Breakpoint 1 is encountered again; DELTA displays the breakpoint message and the instruction. The contents of R0 are examined (/ command) and program execution continues (;P).

⓭ Breakpoint 2 is encountered again; DELTA displays the breakpoint message and the instruction. The subroutine is stepped over again with the O command. The subroutine is executed, and the output is displayed. The instruction where the subroutine returns control is displayed. Program execution continues (;P command).

⓮ Breakpoint 1 is encountered; DELTA displays the breakpoint message and the instruction.

⓯ All breakpoints in the program are listed with the ;B command.

⓰ DELTA displays the breakpoints (by breakpoint number) and the address locations.

⓱ Breakpoint 1 is cleared using 0,[breakpoint #];B. (Never clear breakpoint 1 in XDELTA.)

⓲ All breakpoints are listed again with ;B command.

⓳ DELTA displays breakpoint 2 (breakpoint 1 cleared).

⓴ Program execution continues using the ;P command.

㉑ Breakpoint 2 is encountered; DELTA displays the breakpoint message and the instruction. The subroutine is executed with the O command and the subroutine output is displayed. The next instruction where the subroutine returns control is displayed. Program execution continues with the ;P command.

㉒ Breakpoint 2 is encountered; DELTA displays the breakpoint message and the instruction. Program execution continues to the next breakpoint with the ;P command. The subroutine is executed, and the subroutine output is displayed.

㉓ Breakpoint 2 is encountered again; the instruction at offset 4B9 (in the subroutine) is displayed using !. This instruction is part of the setup for the call to the system service $QIOW.

㉔ Successive address locations are displayed by pressing the Linefeed key nine times. These instructions are the remainder of the setup and the call to the system service $QIOW.

㉕ A breakpoint at X1+04D5 (the current address) is set using the ;B command. This breakpoint is in the subroutine. The . symbol represents the current address.

㉖ The current breakpoints in the program are listed. The new breakpoint is assigned breakpoint 1.

㉗ Program execution continues with the ;P command.

㉘ Program execution stops at the new breakpoint 1, which is in the subroutine GOT_IT. DELTA displays the breakpoint message and the instruction at the new breakpoint. Program execution continues with the ;P command.

㉙ The subroutine completes and displays the output, and program execution continues until breakpoint 2. DELTA displays the breakpoint message and the breakpoint 2 instruction. Program execution continues with the ;P command.

㉚ Program execution stops at breakpoint 1 in the subroutine. Program execution continues with the ;P command. The subroutine is executed, and the output is displayed.

㉛ Program execution stops at breakpoint 2. The O command is entered to execute and step over the subroutine.

㉜ Program execution stops at breakpoint 1 in the subroutine. Program execution continues with the ;P command.

㉝ The subroutine completes execution and displays output. DELTA displays a STEPOVER break message to state that the O command has been completed, returning control at address 69B (an instruction in the main routine).

㉞ The instruction where the subroutine returns is displayed, and program execution is halted. The ;B command is entered to display all current breakpoints.

㉟ The two current breakpoints are listed.

㊱ The command 0,2;B clears breakpoint 2.

㊲ The command 0,1;B clears breakpoint 1.

㊳ The ;B command is entered to display all current breakpoints. Because all breakpoints have been cleared, DELTA does not display any.

㊴ Program execution continues with the ;P command. Because there are no longer any breakpoints, the program executes to the end.

㊵ All current process login times are displayed.

㊶ Final exit status is displayed.

㊷ The DELTA EXIT command is entered to terminate the debugging session and leave DELTA.

B

Sample DELTA Debug Session on AXP

This appendix gives an example of using DELTA to debug a program on OpenVMS AXP. The C program named LOG uses the system service SYS$GETJPIW to obtain the PID, process name, and login time of each process. Although this is an example of using DELTA, most of the commands in the example could be used in an XDELTA debugging session.

To run this program without error, you need WORLD privilege.

The listing file for LOG is shown in two parts. The C source code part is shown in Example B–1. The machine code part is shown in Example B–2.

Example B–1 Listing File for LOG: C Source Code

```
1    #include <descrip.h>
434  #include <jpidef.h>
581  #include <ssdef.h>
1233 #include <starlet.h>
3784 #include <stdio.h>
4117 #include <stdlib.h>
4345
4346 void print_line(unsigned long int pid, char *process_name,
4347   unsigned long int *time_buffer);
4348
4349 typedef struct {
4350   unsigned short int il3_buffer_len;
4351   unsigned short int il3_item_code;
4352   void *il3_buffer_ptr;
4353   unsigned short int *il3_return_len_ptr;
4354         } item_list_3;
4355
4356 #define NUL '\0'
4357
4358 main()
4359   {
4360  static char name_buf[16];
4361  static unsigned long int pid, time_buf[2];
4362  static unsigned short int name_len;
4363
4364  unsigned short int pidadr[2] = {-1, -1};
4365  unsigned long int ss_sts;
4366  item_list_3 jpi_itmlst[] = {
4367          /* Get's login time */
4368          {sizeof(time_buf),
4369           JPI$_LOGINTIM,
4370           (void *) time_buf,
4371           NULL},
4372
4373          /* Get's process name */
```

(continued on next page)

Example B–1 (Cont.) Listing File for LOG: C Source Code

```
4374          {sizeof(name_buf) - 1,
4375           JPI$_PRCNAM,
4376           (void *) name_buf,
4377           &name_len},
4378
4379          /* Get's process ID (PID) */
4380          {sizeof(pid),
4381           JPI$_PID,
4382           (void *) &pid,
4383           NULL},
4384
4385          /* End of list */
4386          {0,
4387           0,
4388           NULL,
4389           NULL}
4390         };
4391
4392 /*
4393 While there's more GETJPI information to process and a catastrophic
4394 error has not occurred then
4395     If GETJPI was successful then
4396         NUL terminate the process name string and
4397  print the information returned by GETJPI
4398 */
4399
4400  while(
4401           (ss_sts = sys$getjpiw(0, &pidadr, 0, &jpi_itmlst, 0, 0, 0)) != SS$_NOMOREPROC &&
4402    ss_sts != SS$_BADPARAM &&
4403    ss_sts != SS$_ACCVIO)
4404    {
4405   if (ss_sts == SS$_NORMAL)
4406     {
4407    *(name_buf + name_len) = NUL;
4408    print_line(pid, name_buf, time_buf);
4409     }
4410    }
4411  exit(EXIT_SUCCESS);
4412   }
4413
4414 void print_line(unsigned long int pid, char *process_name,
4415   unsigned long int *time_buffer)
4416   {
4417  static char ascii_time[12];
4418
4419  struct dsc$descriptor_s time_dsc = {
4420           sizeof(ascii_time) - 1,
4421                  DSC$K_DTYPE_T,
4422                  DSC$K_CLASS_S,
4423                  ascii_time
4424              };
4425  unsigned short int time_len;
4426
4427 /*
4428 Convert the logged in time to ASCII and NUL terminate it
4429 */
4430  sys$asctim(&time_len, &time_dsc, time_buffer, 1);
4431  *(ascii_time + time_len) = NUL;
4432
```

(continued on next page)

Example B–1 (Cont.) Listing File for LOG: C Source Code

```
4433 /*
4434 Output the PID, process name and logged in time
4435 */
4436  printf("\n\tPID= %08.8X\t\tPRCNAM= %s\tLOGINTIM= %s", pid,
4437    process_name, ascii_time);
4438
4439  return;
4440   }
4441 __main(void *p1, void *p2, void *p3, void *p4, void *p5, void *p6)
4442 {
4443     void decc$exit(int);
4444     void decc$main(void *, void *, void *, void *, void *, void *, int *, void **, void **);
4445     int status;
4446     int argc;
4447     void *argv;
4448     void *envp;
4449
4450     decc$main(p1, p2, p3, p4, p5, p6, &argc, &argv, &envp);
4451
4452     status = main
4453                 (
4454
4455
4456
4457                 );
4458
4459     decc$exit(status);
4460 }
```

Example B–2 Listing File for LOG: Machine Code

```
                        .PSECT  $CODE, OCTA, PIC, CON, REL, LCL, SHR,-
                                EXE, NORD, NOWRT
0000    print_line::                                                ; 004414
0000            LDA     SP, -80(SP)             ; SP, -80(SP)
0004            MOV     1, R19                  ; 1, R19            ; 004430
0008            STQ     R27, (SP)               ; R27, (SP)         ; 004414
000C            MOV     4, R25                  ; 4, R25            ; 004430
0010            STQ     R26, 32(SP)             ; R26, 32(SP)       ; 004414
0014            STQ     R2, 40(SP)              ; R2, 40(SP)
0018            STQ     R3, 48(SP)              ; R3, 48(SP)
001C            STQ     R4, 56(SP)              ; R4, 56(SP)
0020            STQ     FP, 64(SP)              ; FP, 64(SP)
0024            MOV     SP, FP                  ; SP, FP
0028            MOV     R27, R2                 ; R27, R2
002C            STL     R17, process_name       ; R17, 16(FP)
0030            LDQ     R0, 40(R2)              ; R0, 40(R2)        ; 004419
0034            MOV     R16, pid                ; R16, R3           ; 004414
0038            LDQ     R26, 48(R2)             ; R26, 48(R2)       ; 004430
003C            LDA     R16, time_len           ; R16, 8(FP)
```

(continued on next page)

Example B–2 (Cont.) Listing File for LOG: Machine Code

```
0040            LDQ     R4, 32(R2)              ; R4, 32(R2)    ; 004423
0044            LDA     R17, time_dsc           ; R17, 24(FP)   ; 004430
0048            STQ     R0, time_dsc            ; R0, 24(FP)    ; 004419
004C            LDQ     R27, 56(R2)             ; R27, 56(R2)   ; 004430
0050            STL     R4, 28(FP)              ; R4, 28(FP)    ; 004419
0054            JSR     R26, SYS$ASCTIM         ; R26, R26      ; 004430
0058            LDL     R0, time_len            ; R0, 8(FP)     ; 004431
005C            MOV     pid, R17                ; R3, R17       ; 004436
0060            LDQ     R27, 88(R2)             ; R27, 88(R2)
0064            MOV     R4, R19                 ; R4, R19
0068            LDQ     R26, 80(R2)             ; R26, 80(R2)
006C            MOV     4, R25                  ; 4, R25
0070            ZEXTW   R0, R0                  ; R0, R0        ; 004431
0074            ADDQ    R4, R0, R0              ; R4, R0, R0
0078            LDQ_U   R16, (R0)               ; R16, (R0)
007C            MSKBL   R16, R0, R16            ; R16, R0, R16
0080            STQ_U   R16, (R0)               ; R16, (R0)
0084            LDQ     R16, 64(R2)             ; R16, 64(R2)   ; 004436
0088            LDL     R18, process_name       ; R18, 16(FP)
008C            JSR     R26, DECC$GPRINTF       ; R26, R26
0090            MOV     FP, SP                  ; FP, SP        ; 004439
0094            LDQ     R28, 32(FP)             ; R28, 32(FP)
0098            LDQ     R2, 40(FP)              ; R2, 40(FP)
009C            LDQ     R3, 48(FP)              ; R3, 48(FP)
00A0            LDQ     R4, 56(FP)              ; R4, 56(FP)
00A4            LDQ     FP, 64(FP)              ; FP, 64(FP)
00A8            LDA     SP, 80(SP)              ; SP, 80(SP)
00AC            RET     R28                     ; R28

Routine Size: 176 bytes,    Routine Base: $CODE + 0000
00B0     main::                                                 ; 004358
00B0            LDA     SP, -144(SP)            ; SP, -144(SP)
00B4            MOV     48, R17                 ; 48, R17       ; 004366
00B8            STQ     R27, (SP)               ; R27, (SP)     ; 004358
00BC            STQ     R26, 64(SP)             ; R26, 64(SP)
00C0            STQ     R2, 72(SP)              ; R2, 72(SP)
00C4            STQ     R3, 80(SP)              ; R3, 80(SP)
00C8            STQ     R4, 88(SP)              ; R4, 88(SP)
00CC            STQ     R5, 96(SP)              ; R5, 96(SP)
00D0            STQ     R6, 104(SP)             ; R6, 104(SP)
00D4            STQ     R7, 112(SP)             ; R7, 112(SP)
00D8            STQ     R8, 120(SP)             ; R8, 120(SP)
00DC            STQ     FP, 128(SP)             ; FP, 128(SP)
00E0            MOV     SP, FP                  ; SP, FP
00E4            MOV     R27, R2                 ; R27, R2
00E8            LDA     SP, -16(SP)             ; SP, -16(SP)
00EC            LDQ     R26, 40(R2)             ; R26, 40(R2)   ; 004366
00F0            LDQ     R18, 64(R2)             ; R18, 64(R2)
00F4            LDA     R16, jpi_itmlst         ; R16, 16(FP)
00F8            JSR     R26, OTS$MOVE           ; R26, R26
00FC            LDA     R6, jpi_itmlst          ; R6, 16(FP)    ; 004401
0100            LDQ     R3, -64(R2)             ; R3, -64(R2)   ; 004370
0104            LDA     R7, pidadr              ; R7, 8(FP)     ; 004401
0108            LDQ     R0, 32(R2)              ; R0, 32(R2)    ; 004364
010C            MOV     2472, R8                ; 2472, R8      ; 004401
0110            STL     R0, pidadr              ; R0, 8(FP)     ; 004364
0114            LDA     R3, time_buf            ; R3, 16(R3)    ; 004370
```

(continued on next page)

Example B–2 (Cont.) Listing File for LOG: Machine Code

```
0118            MOV     R3, R5                  ; R3, R5
011C            STL     R5, 20(FP)              ; R5, 20(FP)    ; 004366
0120            LDA     R4, 8(R3)               ; R4, 8(R3)     ; 004376
0124            STL     R4, 32(FP)              ; R4, 32(FP)    ; 004366
0128            LDA     R17, 24(R3)             ; R17, 24(R3)
012C            STL     R17, 36(FP)             ; R17, 36(FP)
0130            LDA     R19, 28(R3)             ; R19, 28(R3)
0134            STL     R19, 44(FP)             ; R19, 44(FP)
0138    L$6:                                                    ; 004400
0138            LDQ     R26, 48(R2)             ; R26, 48(R2)   ; 004401
013C            CLR     R16                     ; R16
0140            LDQ     R27, 56(R2)             ; R27, 56(R2)
0144            MOV     R7,  R17                ; R7, R17
0148            STQ     R31, (SP)               ; R31, (SP)
014C            CLR     R18                     ; R18
0150            MOV     R6, R19                 ; R6, R19
0154            CLR     R20                     ; R20
0158            CLR     R21                     ; R21
015C            MOV     7, R25                  ; 7, R25
0160            JSR     R26, SYS$GETJPIW        ; R26, R26
0164            CMPEQ   ss_sts, 20, R16         ; R0, 20, R16   ; 004402
0168            CMPEQ   ss_sts, R8, R17         ; R0, R8, R17   ; 004401
016C            CMPEQ   ss_sts, 12, R18         ; R0, 12, R18   ; 004403
0170            BIS     R17, R16, R17           ; R17, R16, R17 ; 004401
0174            BIS     R17, R18, R18           ; R17, R18, R18
0178            BNE     R18, L$10               ; R18, L$10     ; 004400
017C            CMPEQ   ss_sts, 1, R0           ; R0, 1, R0     ; 004405
0180            BEQ     R0, L$6                 ; R0, L$6
0184            MOV     R4, R17                 ; R4, R17       ; 004408
0188            LDQ_U   R19, 24(R3)             ; R19, 24(R3)   ; 004407
018C            MOV     R5, R18                 ; R5, R18       ; 004408
0190            LDA     R27, -96(R2)            ; R27, -96(R2)
0194            EXTWL   R19, R3, R19            ; R19, R3, R19  ; 004407
0198            ADDQ    R4, R19, R19            ; R4, R19, R19
019C            LDQ_U   R22, (R19)              ; R22, (R19)
01A0            MSKBL   R22, R19, R22           ; R22, R19, R22
01A4            STQ_U   R22, (R19)              ; R22, (R19)
01A8            LDL     R16, 28(R3)             ; R16, 28(R3)   ; 004408
01AC            BSR     R26, print_line         ; R26, print_line
01B0            BR      L$6                     ; L$6           ; 004405
01B4            NOP                             ;
01B8    L$10:                                                   ; 004400
01B8            LDQ     R26, 80(R2)             ; R26, 80(R2)   ; 004411
01BC            CLR     R16                     ; R16
01C0            LDQ     R27, 88(R2)             ; R27, 88(R2)
01C4            MOV     1, R25                  ; 1, R25
01C8            JSR     R26, DECC$EXIT          ; R26, R26
01CC            MOV     FP, SP                  ; FP, SP        ; 004412
01D0            LDQ     R28, 64(FP)             ; R28, 64(FP)
01D4            MOV     1, R0                   ; 1, R0
01D8            LDQ     R2, 72(FP)              ; R2, 72(FP)
01DC            LDQ     R3, 80(FP)              ; R3, 80(FP)
01E0            LDQ     R4, 88(FP)              ; R4, 88(FP)
01E4            LDQ     R5, 96(FP)              ; R5, 96(FP)
```

(continued on next page)

Example B–2 (Cont.) Listing File for LOG: Machine Code

```
01E8            LDQ     R6, 104(FP)             ; R6, 104(FP)
01EC            LDQ     R7, 112(FP)             ; R7, 112(FP)
01F0            LDQ     R8, 120(FP)             ; R8, 120(FP)
01F4            LDQ     FP, 128(FP)             ; FP, 128(FP)
01F8            LDA     SP, 144(SP)             ; SP, 144(SP)
01FC            RET     R28                     ; R28

Routine Size: 336 bytes,    Routine Base: $CODE + 00B0
0200    __main::                                                ; 004441
0200            LDA     SP, -48(SP)             ; SP, -48(SP)
0204            MOV     9, R25                  ; 9, R25        ; 004450
0208            STQ     R27, (SP)               ; R27, (SP)     ; 004441
020C            STQ     R26, 24(SP)             ; R26, 24(SP)
0210            STQ     R2, 32(SP)              ; R2, 32(SP)
0214            STQ     FP, 40(SP)              ; FP, 40(SP)
0218            MOV     SP, FP                  ; SP, FP
021C            LDA     SP, -32(SP)             ; SP, -32(SP)
0220            MOV     R27, R2                 ; R27, R2
0224            LDA     R0, argc                ; R0, 16(FP) ; 004450
0228            LDQ     R26, 48(R2)             ; R26, 48(R2)
022C            LDA     R1, argv                ; R1, 12(FP)
0230            STQ     R0, (SP)                ; R0, (SP)
0234            LDA     R0, envp                ; R0, 8(FP)
0238            STQ     R1, 8(SP)               ; R1, 8(SP)
023C            LDQ     R27, 56(R2)             ; R27, 56(R2)
0240            STQ     R0, 16(SP)              ; R0, 16(SP)
0244            JSR     R26, DECC$MAIN          ; R26, R26
0248            LDA     R27, -96(R2)            ; R27, -96(R2) ; 004452
024C            BSR     R26, main               ; R26, main
0250            LDQ     R27, 40(R2)             ; R27, 40(R2)   ; 004459
0254            MOV     status, R16             ; R0, R16
0258            MOV     1, R25                  ; 1, R25
025C            LDQ     R26, 32(R2)             ; R26, 32(R2)
0260            JSR     R26, DECC$EXIT          ; R26, R26
0264            MOV     FP, SP                  ; FP, SP        ; 004460
0268            LDQ     R28, 24(FP)             ; R28, 24(FP)
026C            LDQ     R2, 32(FP)              ; R2, 32(FP)
0270            LDQ     FP, 40(FP)              ; FP, 40(FP)
0274            LDA     SP, 48(SP)              ; SP, 48(SP)
0278            RET     R28                     ; R28

Routine Size: 124 bytes,    Routine Base: $CODE + 0200
```

The .MAP file for the sample program is shown in Example B–3. Only the Program Section Synopsis with the psect, module, base address, end address, and length are listed.

The DELTA debug session is shown in Example B–4.

Example B–3 .MAP File for the Sample Program

```
                                        +--------------------------+
                                        ! Program Section Synopsis !
                                        +--------------------------+

Psect Name      Module Name      Base     End          Length
----------      -----------      ----     ---          ------

$LINKAGE                         00010000 000100FF 00000100 (       256.)
                LOG              00010000 000100FF 00000100 (       256.)
$LITERAL                         00010100 00010158 00000059 (        89.)
                LOG              00010100 00010158 00000059 (        89.)
$READONLY                        00010160 00010160 00000000 (         0.)
                LOG              00010160 00010160 00000000 (         0.)
$INIT                            00020000 00020000 00000000 (         0.)
                LOG              00020000 00020000 00000000 (         0.)
$UNINIT                          00020000 0002002F 00000030 (        48.)
                LOG              00020000 0002002F 00000030 (        48.)
$CODE                            00030000 0003027B 0000027C (       636.)
                LOG              00030000 0003027B 0000027C (       636.)
```

Example B–4 DELTA Debugging Session of the Sample Program

```
$ DEFINE LIB$DEBUG SYS$LIBRARY:DELTA ❶
$ RUN/DEBUG LOG ❷
Alpha/VMS DELTA Version 1.5 ❸

Brk 0 at 00030200

00030200!    LDA         SP,#XFFD0(SP)  30000,1;X
X1 164! CMPEQ           R0,#X14,R16  .;B ❹

X1 1AC! BSR             R26,#XFFFF94  .;B ❺

;P

Brk 1 at 00030164 ❻

X1+00000164!    CMPEQ           R0,#X14,R16  R0/ 00000001 ;P

Brk 2 at 000301AC

X1+000001AC!    BSR             R26,#XFFFF94  O

       PID= 00000021            PRCNAM= SWAPPER LOGINTIM= 00:00:00.00 ❼
X1+000001B0!    BR              R31,#XFFFFE1  ;P

Brk 1 at 00030164

X1+00000164!    CMPEQ           R0,#X14,R16  R0/ 00000001 ;P
```

(continued on next page)

Example B–4 (Cont.) DELTA Debugging Session of the Sample Program

```
Brk 2 at 000301AC

X1+000001AC!    BSR             R26,#XFFFF94  O (8)
       PID= 00000024            PRCNAM= ERRFMT  LOGINTIM= 16:24:01.03
X1+000001B0!    BR              R31,#XFFFFE1  ;P

Brk 1 at 00030164

X1+00000164!    CMPEQ           R0,#X14,R16

;B
 1 00030164
 2 000301AC

0,1;B

;B
 2 000301AC

;P

Brk 2 at 000301AC  (9)

X1+000001AC!    BSR             R26,#XFFFF94  O
       PID= 00000025            PRCNAM= OPCOM   LOGINTIM= 16:24:02.56
X1+000001B0!    BR              R31,#XFFFFE1  ;P

Brk 2 at 000301AC  (10)

X1+000001AC!    BSR             R26,#XFFFF94  O
     PID= 00000026          PRCNAM= AUDIT_SERVER    LOGINTIM=16:24:03.66
X1+000001B0!    BR              R31,#XFFFFE1  ;P

Brk 2 at 000301AC  (11)

X1+000001AC!    BSR         R26,#XFFFF94 X1 84!  LDQ   R16,#X0040(R2)
Linefeed  (12)

X1+00000088!    LDL             R18,#X0010(FP)  Linefeed

X1+0000008C!    JSR             R26,(R26)  .;B (13)

;B
 1 0003008C
 2 000301AC

;P (14)

Brk 1 at 0003008C  (15)

X1+0000008C!    JSR             R26,(R26)  O
     PID= 00000027          PRCNAM= JOB_CONTROL     LOGINTIM= 16:24:06.83
X1+00000090!    BIS             R31,FP,SP  ;P

Brk 2 at 000301AC

X1+000001AC!    BSR             R26,#XFFFF94  ;P
Brk 1 at 0003008C (16)

X1+0000008C!    JSR             R26,(R26)  O
     PID= 00000028          PRCNAM= NETACP  LOGINTIM= 16:24:22.86
X1+00000090!    BIS             R31,FP,SP  ;P

Brk 2 at 000301AC

X1+000001AC!    BSR             R26,#XFFFF94
```

(continued on next page)

Example B–4 (Cont.) DELTA Debugging Session of the Sample Program

```
;B
 1 0003008C
 2 000301AC

0,2;B

0,1;B

;B

;P

        PID= 00000029        PRCNAM= EVL     LOGINTIM= 16:24:26.67
        PID= 0000002A        PRCNAM= REMACP  LOGINTIM= 16:24:38.21
        PID= 0000002B        PRCNAM= LATACP  LOGINTIM= 16:24:43.18
        PID= 0000004C        PRCNAM= GODDARD LOGINTIM= 07:40:49.34
        PID= 0000002D        PRCNAM= SYMBIONT_0001   LOGINTIM= 16:25:47.54
        PID= 0000002F        PRCNAM= MCCORMICK       LOGINTIM= 16:27:45.27
Exit 00000001

8002228C!       ADDL            R15,SP,SP EXIT
```

❶ DELTA is enabled as the debugger.

❷ The example program Log is invoked with DELTA.

❸ DELTA displays a version number and the first executable instruction. The base address of the program (determined from the map file) is virtual address 30000. The base address is placed in base register 1 with ;X. Now references to an address can use the address offset notation. For example, a reference to the first instruction is X1+200 (or the base address 30000 + offset 200). Also, DELTA displays some address locations as offsets to the base address.

❹ The instruction at address 30164 is displayed in instruction mode using !. Its address location is expressed as the base address plus an offset. In the listing file, the offset is 164. (This is the point where the return status from SYS$GETJPIW is checked.) The base address in base address register X1 is 30000. The address reference, then, is X1+164. Note the + sign is implied when not specified.

A simple breakpoint is set at that address using the ;B command. The address reference for ;B is the . symbol, representing the current address. X1+164;B would have done the same thing.

❺ The same commands (! command to view the instruction and ;B to set a breakpoint) are repeated for the instruction at offset 1AC. (This is the point at which the print_line function is called.)

❻ Program execution halts at the first breakpoint. DELTA displays the breakpoint message (Brk 1 at 00030164) with the breakpoint number 1 and the virtual address. The virtual address is 30164, which is the base address (30000) plus the offset 164. DELTA then displays the instruction in instruction mode (CMPEQ R0,#X14,R16). The contents of the general register 0 are displayed with the / command. DELTA displays the contents of R0, which is 1. Program execution continues using the ;P command.

❼ The function print_line is executed, and the output (PID, process name, and login time) is displayed.

❽ The O command halts program execution at the instruction where the function returns control (BR R31,#XFFFFE1). (This is the point at which control passes to checking the conditions of the while loop.) Program execution continues with ;P.

❾ Breakpoint 2 is encountered. DELTA displays the breakpoint message, and the instruction. The function is executed with the O command and the function output is displayed. The next instruction where the function returns control is displayed. Program execution continues with the ;P command.

❿ Breakpoint 2 is encountered again. DELTA displays the breakpoint message, and the instruction. The function is executed with the O command and the function output is displayed. The next instruction where the function returns control is displayed. Program execution continues with the ;P command.

⓫ Breakpoint 2 is encountered again. The instruction at offset 84 (in print_line) is displayed using !. This instruction is part of the setup for the call to the printf function.

⓬ Successive address locations are displayed by pressing the Linefeed key two times. These instructions are the remainder of the setup and the call to printf.

⓭ A breakpoint at X1+8C (the current address) is set using the ;B command. This breakpoint is in the function print_line. The . symbol represents the current address. Note that breakpoint 1 was cleared earlier and is now reused by DELTA for the new breakpoint.

⓮ Program execution continues with the ;P command.

⓯ Program execution stops at the new breakpoint 1, which is in the print_line function. DELTA displays the breakpoint message and the instruction at the new breakpoint. The O command halts program execution at the instruction where the function returns control, stepping over the routine call. Note the O command must be used in this case, as opposed to the ;P command, because the printf function resides in read-only protected memory. Program execution is continued with the ;P command.

⓰ Program execution stops at breakpoint 1 in the print_line function. Program execution is continued using a combination of the O and ;P commands.

Index

A

Address location
  changing the value, 4–5
  closing current, 4–9, 4–15
  command strings (XDELTA), 2–1, 4–27
  displaying contents of current, 4–4
  displaying in ASCII, 4–13
  displaying location pointed to by current location, 4–12
  displaying next, 4–9
  displaying previous, 4–11
  displaying range, 4–4
  displaying, from other processes, 4–4
  listing for executive images, 4–33
  PCB, 2–1
  referencing, 3–1
  using base address and offsets, 3–2, 3–4
Address symbol
  current, 2–1
Arithmetic operators, 2–2
Arithmetic shift, 2–3
ASCII
  depositing string, 4–26
  displaying contents, 4–13

B

;B command, 4–16
Base register
  loading, 4–29
  symbol, 2–1
Boot command
  qualifiers for local and HSC disks, 1–8
  qualifiers for XDELTA, 1–3
  selection of root, 1–8
Boot procedures for XDELTA, 1–4
  See also entries for individual computers
Breakpoint
  accessing initial on AXP, 1–13
  accessing initial on VAX, 1–13
  clearing, 4–16, 4–18
  complex, 4–18
  initial in multiprocessor environment, 1–14
  initial in XDELTA, 1–13
  multiprocessor environment, 1–14, 3–9, 4–23
  proceeding from, 4–19
Breakpoint (cont'd)
  proceeding from initial, 1–13
  range for DELTA, 4–16
  range for XDELTA, 4–16
  setting, 4–16, 4–17
  showing, 4–16
  simple, 4–17
  XDELTA restriction on breakpoint 1, 1–11
Breakpoint command, 4–16
Bugcheck information, 1–14

C

;C command, 4–36
Close Current Location, Open Next command, 4–9
Code pages
  making writeable, 3–7
' command, 4–26
= command, 4–31
[command, 4–3
/ command, 4–4
" command, 4–13
Command list, 4–1
Complex breakpoint, 4–18
Control region space prefix symbol, 2–2
Copy-on-reference (CRF), 3–8
CPU ID, 4–17, 4–23
Crash command, 4–36
CRF (copy-on-reference), 3–8

D

dddXDT
  XDELTA boot, 1–5
Debugging
  at elevated IPL, 1–1
  at IPL 0, 1–1
  code that does not match compiler listing, 3–10
  kernel mode code in process space, 3–7
  privileged code, 1–1
  user-mode programs, 1–1
Delta/XDelta Debugger
  exiting from DELTA, 1–2
  exiting from XDELTA, 1–14
  invoking DELTA, 1–2
  invoking XDELTA, 1–2

Delta/XDelta utility
debugging an installed, protected, shareable image, 3–8
Deposit ASCII String command, 4–26
Display information commands
See List commands
Display mode
how to set, 4–3
Display Value of Expression command, 4–31

E

;E command, 4–27
Eh? error message, 3–7
ESC command, 4–11
ESC key equivalent, 4–11
Evaluation precedence, 2–2
Exclamation Point (!) command, 4–7
Execute Command String command, 4–27
Executive images
listing names and addresses, 4–33, 4–37
Exit command, 4–43
Exiting
from DELTA, 1–2, 4–43
from XDELTA, 1–14
Expressions
See also Numeric expressions
precedence in, 2–2

F

Floating point control register (FPCR), 2–2
Floating point registers, 2–2
FPCR symbol, 2–2

G

;G command, 4–20
G symbol, 2–2
General register symbol, 2–2, 3–6
Go command, 4–20

H

;H command, 4–41
H symbol, 2–2
Hardcopy output command, 4–41

I

;I command, 4–39
Image code
does not match compiler listing, 3–10
Images, sliced, 4–33, 4–37, 4–39
Immediate mode text display command, 4–42
INI$BRK routine, 1–11, 3–8, 4–17
AXP, 1–13
VAX, 1–13
Initial breakpoint
See Breakpoint
Instructions
how to display, 4–7
Internal processor register, 2–2
Interrupt request for XDELTA, 1–11
See also entries for individual computers
Invoking
See also Boot procedures for XDELTA
See also Interrupt request for XDELTA
DELTA, 1–2
XDELTA, 1–2
IPID, 2–2, 4–4, 4–7, 4–13
IPR symbolic names
Alpha AXP, 2–2

K

Kernel mode code in process space
debugging, 3–7

L

;L command, 4–33
privileges required, 1–1
LINEFEED command, 4–9
Linefeed key equivalent, 4–9
Linker options file
used with XDELTA, 3–7
LIS file, 3–1, 3–3
List commands
Information About the Current Main Image and Its Shareable Images, 4–39
Name and Location of a Single Image, 4–37
Names and Addresses of Loaded Executive Images, 4–33
Load Base Register command, 4–29

M

;M command, 4–32
privileges required, 1–1
MAP file, 3–1, 3–2, 3–3
MicroVAX 2000
boot procedure for XDELTA, 1–9
requesting interrupt, 1–12
MicroVAX 3100 series
boot procedure for XDELTA, 1–8
MicroVAX 3300/3400 series
boot procedure for XDELTA, 1–9
requesting interrupt, 1–12
MicroVAX 3500/3600 series
boot procedure for XDELTA, 1–9
requesting interrupt, 1–12
MicroVAX 3800/3900 series
boot procedure for XDELTA, 1–9
requesting interrupt, 1–12

MicroVAX II
  boot procedure for XDELTA, 1–9
  requesting interrupt, 1–12
Multiprocessor environment
  initial breakpoint, 1–14
  XDELTA breakpoints, 1–14, 3–9, 4–17, 4–23
  XDELTA operation, 3–9

N

Numeric expressions, 2–2, 4–31

O

O command, 4–23
Open Location and Display Contents command, 4–4
Open Location and Display Contents in Instruction Mode command, 4–7
Open Location and Display Indirect Location command, 4–12
Open Location and Display Previous Location command, 4–11
Operators
  arithmetic, 2–2
Output
  from DELTA, 1–1
  from XDELTA, 1–2

P

;P command, 4–19
Page faults
  preventing, 3–7
PCB address location, 2–1
PCB vector start symbolic address, 2–1
PFN (physical page number), 4–27
PID
  See IPID
Pn symbol, 2–2
Printed output command, 4–41
Privileges
  DELTA, 1–1
  XDELTA, 1–2
Proceed from Breakpoint command, 4–19
Processes
  how to set writable, 4–32
Processor register symbol, 2–2
Processor status longword symbol
  See PSL
Processor status symbol
  See PS
Program execution
  continuing, 4–20
  proceeding from breakpoint, 4–19
  step execution, 4–21
  step over subroutine execution, 4–23
PS (processor status symbol), 2–2, 3–6
PSL (processor status longword), 2–2, 3–6

Q

;Q command, 4–35
Q symbol, 2–1
Queue
  validate, 4–35

R

R3 qualifier in BOOT command, 1–8
R5 qualifier in BOOT command, 1–8
Radix, 2–1
Redirecting output
  DELTA, 1–1
  XDELTA, 1–2
Registers
  display contents, 4–4
  examining general purpose registers of another process, 4–6
  loading base, 4–29
  symbol for base, 2–1
  symbol for general, 3–6
  symbol for processor, 2–2
RETURN command, 4–15
Rn symbol, 2–2

S

S command, 4–21
SCH$GL_PCBVEC symbolic address, 2–1
Set All Processes Writable command, 4–32
Set Display Mode command, 4–3
$SETPRT
  used with XDELTA, 3–7
Shareable images
  debugging installed, protected, 3–8
  list information about current main and, 4–39
Simple breakpoint, 4–17
Single-step
  fails, 3–9
Sliced images, 4–33, 4–37, 4–39
Stack pointer symbol, 2–2, 3–6
Step Instruction command, 4–21
Step Instruction over Subroutine command, 4–23
String
  depositing ASCII, 4–26
string\ command, 4–42
. symbol, 2–1
Symbol list, 2–1
System space prefix symbol, 2–2

T

TAB command, 4–12
Terminating DELTA, 4–43
Terminating DELTA/XDELTA commands, 4–15
TU58 console boot procedures, 1–10

V

Validate queue command, 4–35
Value (last) displayed symbol, 2–1
VAX 6000 series
  boot procedure for XDELTA, 1–7
  requesting interrupt, 1–11
VAXstation 3100 series
  boot procedure for XDELTA, 1–8
VAXstation 4000 series
  boot procedure for XDELTA, 1–8
VAX–11/730
  boot procedure for XDELTA, 1–10
  requesting interrupt, 1–11
VAX–11/750
  boot procedure for XDELTA, 1–9
    with TU58 console, 1–10
  requesting interrupt, 1–11
VAX–11/780
  boot procedure for XDELTA, 1–9
  requesting interrupt, 1–11
VAX–11/785
  boot procedure for XDELTA, 1–9
  requesting interrupt, 1–11
VAX 10000 series
  boot procedure for XDELTA, 1–7
  requesting interrupt, 1–12
VAX 4000 series
  boostrap procedure for XDELTA, 1–9
  requesting interrupt, 1–12
VAX 7000 series
  boot procedure for XDELTA, 1–7
  requesting interrupt, 1–12
VAX 8200
  boot procedure for XDELTA, 1–6
  requesting interrupt, 1–11
VAX 8250
  boot procedure for XDELTA, 1–6
  requesting interrupt, 1–11
VAX 8300
  boot procedure for XDELTA, 1–6
  requesting interrupt, 1–11
VAX 8350
  boot procedure for XDELTA, 1–6
  requesting interrupt, 1–11
VAX 8530
  boot procedure for XDELTA, 1–5
  requesting interrupt, 1–11
VAX 8550
  boot procedure for XDELTA, 1–5
  requesting interrupt, 1–11
VAX 8600
  boot procedure for XDELTA, 1–6
  requesting interrupt, 1–11
VAX 8650
  boot procedure for XDELTA, 1–6
  requesting interrupt, 1–11
VAX 8700
  See VAX 8810
VAX 8800
  See VAX 8820-N
VAX 8810
  boot procedure for XDELTA, 1–5
  requesting interrupt, 1–11
VAX 8820
  boot procedure for XDELTA, 1–5
  requesting interrupt, 1–11
VAX 8820-N
  boot procedure for XDELTA, 1–5
  requesting interrupt, 1–11
VAX 8830
  boot procedure for XDELTA, 1–5
  requesting interrupt, 1–11
VAX 8840
  boot procedure for XDELTA, 1–5
  requesting interrupt, 1–11
VAX 9000
  boot procedure for XDELTA, 1–4
  requesting interrupt, 1–11
VAXft 3000
  boot procedure for XDELTA, 1–9
  requesting interrupt, 1–12
VAXft-410
  boot procedure for XDELTA, 1–9
  requesting interrupt, 1–12
VAXft-610
  boot procedure for XDELTA, 1–9
  requesting interrupt, 1–12
VAXft-612
  boot procedure for XDELTA, 1–9
  requesting interrupt, 1–12
VAXstation 2000
  boot procedure for XDELTA, 1–9
  requesting interrupt, 1–12
VAXstation 3500/3600 series
  boot procedure for XDELTA, 1–9
  requesting interrupt, 1–12
VAXstation 3520
  boot procedure for XDELTA, 1–9
  requesting interrupt, 1–12
VAXstation 3540
  boot procedure for XDELTA, 1–9
  requesting interrupt, 1–12

Video Terminal Display command, 4–41
Volume shadowing qualifier, 1–8

W

;W command, 4–37

X

;X command, 4–29
X4 symbol, 2–1
X5 symbol, 2–1
XE base register, 2–1, 4–27
XF base register, 2–1, 4–27
Xn symbol, 2–1

NOTES

NOTES

NOTES

NOTES

NOTES

NOTES

NOTES

NOTES

NOTES

NOTES

How to Order Additional Documentation

Technical Support

If you need help deciding which documentation best meets your needs, call 800-DIGITAL (800-344-4825) and press 2 for technical assistance.

Electronic Orders

If you wish to place an order through your account at the Electronic Store, dial 800-234-1998, using a modem set to 2400- or 9600-baud. You must be using a VT terminal or terminal emulator set at 8 bits, no parity. If you need assistance using the Electronic Store, call 800-DIGITAL (800-344-4825) and ask for an Electronic Store specialist.

Telephone and Direct Mail Orders

From	Call	Write
U.S.A.	DECdirect Phone: 800-DIGITAL (800-344-4825) FAX: (603) 884-5597	Digital Equipment Corporation P.O. Box CS2008 Nashua, NH 03061
Puerto Rico	Phone: (809) 781-0505 FAX: (809) 749-8377	Digital Equipment Caribbean, Inc. 3 Digital Plaza, 1st Street Suite 200 Metro Office Park San Juan, Puerto Rico 00920
Canada	Phone: 800-267-6215 FAX: (613) 592-1946	Digital Equipment of Canada Ltd. 100 Herzberg Road Kanata, Ontario, Canada K2K 2A6 Attn: DECdirect Sales
International	———	Local Digital subsidiary or approved distributor
Internal Orders[1] (for software documentation)	DTN: 241-3023 (508) 874-3023	Software Supply Business (SSB) Digital Equipment Corporation 1 Digital Drive Westminster, MA 01473
Internal Orders (for hardware documentation)	DTN: 234-4325 (508) 351-4325 FAX: (508) 351-4467	Publishing & Circulation Services Digital Equipment Corporation NR02-2 444 Whitney Street Northboro, MA 01532

[1]Call to request an Internal Software Order Form (EN–01740–07).

Reader's Comments

OpenVMS Delta/XDelta Debugger Manual
AA–PWCAA–TE

Your comments and suggestions help us improve the quality of our publications.

Thank you for your assistance.

I rate this manual's:	Excellent	Good	Fair	Poor
Accuracy (product works as manual says)	☐	☐	☐	☐
Completeness (enough information)	☐	☐	☐	☐
Clarity (easy to understand)	☐	☐	☐	☐
Organization (structure of subject matter)	☐	☐	☐	☐
Figures (useful)	☐	☐	☐	☐
Examples (useful)	☐	☐	☐	☐
Index (ability to find topic)	☐	☐	☐	☐
Page layout (easy to find information)	☐	☐	☐	☐

I would like to see more/less _____

_____

_____

What I like best about this manual is _____

_____

_____

What I like least about this manual is _____

_____

_____

I found the following errors in this manual:

Page Description

_____ _____

_____ _____

_____ _____

_____ _____

_____ _____

Additional comments or suggestions to improve this manual:

_____

_____

_____

_____

For software manuals, please indicate which version of the software you are using: _____

_____

Name/Title _____ Dept. _____

Company _____ Date _____

Mailing Address _____

_____ Phone _____

---- Do Not Tear - Fold Here and Tape ----

digital™

No Postage Necessary if Mailed in the United States

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL EQUIPMENT CORPORATION
OpenVMS Documentation
110 SPIT BROOK ROAD ZKO3–4/U08
NASHUA, NH 03062–2642

---- Do Not Tear - Fold Here ----

Reader's Comments

OpenVMS Delta/XDelta Debugger Manual
AA–PWCAA–TE

Your comments and suggestions help us improve the quality of our publications.

Thank you for your assistance.

I rate this manual's:	Excellent	Good	Fair	Poor
Accuracy (product works as manual says)	☐	☐	☐	☐
Completeness (enough information)	☐	☐	☐	☐
Clarity (easy to understand)	☐	☐	☐	☐
Organization (structure of subject matter)	☐	☐	☐	☐
Figures (useful)	☐	☐	☐	☐
Examples (useful)	☐	☐	☐	☐
Index (ability to find topic)	☐	☐	☐	☐
Page layout (easy to find information)	☐	☐	☐	☐

I would like to see more/less _____

_____

_____

What I like best about this manual is _____

_____

_____

What I like least about this manual is _____

_____

_____

I found the following errors in this manual:

Page	Description
_____	_____
_____	_____
_____	_____
_____	_____
_____	_____

Additional comments or suggestions to improve this manual:

_____

_____

_____

_____

For software manuals, please indicate which version of the software you are using: _____

_____

Name/Title _____ Dept. _____

Company _____ Date _____

Mailing Address _____

_____ Phone _____

Do Not Tear - Fold Here and Tape

digital™

No Postage Necessary if Mailed in the United States

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL EQUIPMENT CORPORATION
OpenVMS Documentation
110 SPIT BROOK ROAD ZKO3–4/U08

NASHUA, NH 03062–2642

Do Not Tear - Fold Here

digital

Printed in Europe